**Programming in JAVA**

Group Project

**ELECTRICITY BILL MANAGEMENT SYSTEM**

Submitted to:

Dr Surbhi: 23540

Submitted by:

Group – 1

Rohit Prabhakaran (12001550)

Nalle Veera Rama Teja (12001562)

Nakkella Vishnu Vardhan (12002357)

**TABLE OF CONTENT**

1. Introduction
2. Project Description
3. Code Analysis
4. User Interface
5. Testing
6. Conclusion

I. Introduction

The Electricity Bill Management System is a software application designed to simplify the management of electricity bills for residential and commercial customers. The system offers a user-friendly interface that allows customers to view their electricity bills and pay them online, eliminating the need for physical visits to the electricity office.

The system is designed to automate the billing process and reduce the workload of electricity company employees. With the Electricity Bill Management System, electricity companies can easily manage customer information, track bill payments, and generate reports on electricity usage and billing trends.

The purpose of this report is to provide a detailed analysis of the Electricity Bill Management System, including an overview of its features and functionality, an analysis of the Java code, an evaluation of the user interface, and a description of the testing process used to ensure the correctness and robustness of the system.

In this report, we will examine each class in the Java code and analyze their properties and methods, as well as discuss any design patterns used in the code. We will also evaluate the code for clarity, efficiency, and maintainability, and provide suggestions for improvements.

Additionally, we will provide a detailed description of the user interface of the system, including screenshots and explanations of each component. We will also discuss the testing process used to ensure the correctness and robustness of the system, as well as any issues or bugs encountered during testing and their resolution.

Overall, this report aims to provide a comprehensive evaluation of the Electricity Bill Management System, with the goal of identifying its strengths and weaknesses, and suggesting areas for future improvement or enhancement.

II. Project Description

A. Overview of the Electricity Bill Management System

The Electricity Bill Management System is a web-based application developed to simplify the management of electricity bills for both residential and commercial customers. This system automates the billing process and reduces the workload of electricity company employees. With the system's highly scalable design, it can handle a vast amount of customer data and billing information while maintaining robust security features to protect customer data and prevent unauthorized access.

The system offers a user-friendly interface to customers, enabling them to view their electricity bills and pay them online, eliminating the need for physical visits to the electricity office. Customers can also view their payment history, billing information, and update their personal information and contact details.

For electricity companies, the system provides a centralized platform for managing customer information, tracking bill payments, and generating reports on electricity usage and billing trends. The system is capable of generating detailed reports on customer usage patterns, electricity consumption trends, and billing history, which can be used to inform business decisions and improve customer service.

B. Main Features and Functionality of the System

The Electricity Bill Management System offers a wide range of features and functionality that make it easy for customers to manage their electricity bills and for electricity companies to manage customer data and billing information. Some of the main features of the system are:

User-friendly interface: The system provides a simple and intuitive interface that makes it easy for customers to view their bills, update their personal information, and pay their bills online.

Online bill payment: The system enables customers to pay their bills online using various payment methods, including credit cards, debit cards, and net banking.

Automatic bill generation: The system automates the billing process, generating bills automatically and sending them to customers via email or SMS.

Customer data management: The system offers a centralized platform for managing customer data, including personal information, contact details, and billing information.

Billing history and usage reports: The system generates detailed reports on customer usage patterns, electricity consumption trends, and billing history, which can be used to inform business decisions and improve customer service.

Security and privacy features: The system is designed to be highly secure, with robust security features that protect customer data and prevent unauthorized access.

III. Code Analysis

1. Overview

The Customer class: This class represents a customer in the system and contains properties for storing the customer's name, address, and meter readings. It also has methods for getting and setting these properties.

The Bill class: This class represents an electricity bill and contains properties for storing the customer ID, meter readings, and calculated bill amount. It also has methods for calculating the bill amount based on the customer's meter readings.

The ElectricityBillManagementSystem class: This is the main class of the program and contains the main method. It provides a user interface for interacting with the system, allowing users to add, view, and update customer information, as well as generate bills and reports.

In terms of potential issues or areas for improvement, here are a few things to consider:

Data validation: The program does not appear to perform any data validation on user input. This means that users could potentially enter invalid data, which could cause errors or incorrect results. It would be a good idea to add validation checks to ensure that the input is valid before proceeding.

Error handling: The program does not handle errors in a robust way. If an error occurs, the program simply prints an error message and exits. It would be better to handle errors more gracefully, such as by providing informative error messages and allowing the user to correct the error.

Code organization: The code could benefit from better organization and separation of concerns. For example, the Customer and Bill classes could be placed in separate files, and the user interface code could be placed in a separate class.

B. Classes Used in the program.

1. excelRead

The excelRead class contains two static methods: getCustomerDetails() and showAllData().

The getCustomerDetails() method takes two parameters - a file path and a customer ID - and reads data from the Excel sheet. It creates a FileInputStream object to read data from the specified file path, and then creates a XSSFWorkbook object to read the Excel workbook. It then gets the first sheet from the workbook using the getSheetAt() method and creates a FormulaEvaluator object to evaluate formulas in the sheet. A DataFormatter object is also created to format the cell values in the sheet.

The method then loops through each row in the sheet using a for-each loop and retrieves the first cell value in each row using the getCell() method. If the cell contains a numeric value and that value matches the specified customer ID, the method loops through each cell in the row and uses a switch statement to handle each cell value based on its data type. If the cell contains a numeric value, it checks if it is a date value using the isCellDateFormatted() method, and formats the cell value using the formatCellValue() method of the DataFormatter object. If the cell contains a string value, it prints the string value using the getStringCellValue() method. Finally, if the cell contains any other type of data, it prints a tab character.

The showAllData() method also takes a file path as a parameter and reads data from the Excel sheet. It uses the same approach as the getCustomerDetails() method to read the workbook, sheet, and evaluate formulas. It then loops through each row in the sheet and each cell in the row, using a switch statement to handle each cell value based on its data type. If the cell contains a numeric value, it checks if it is a date value using the isCellDateFormatted() method, and formats the cell value using the formatCellValue() method of the DataFormatter object. If the cell contains a string value, it prints the string value using the getStringCellValue() method. Finally, if the cell contains any other type of data, it prints a tab character.

In summary, the excelRead class provides methods for reading data from an Excel sheet. The getCustomerDetails() method allows retrieving data for a specific customer based on their ID, while the showAllData() method allows viewing all the data in the sheet. These methods use the Apache POI library to read and manipulate Excel files.

2. excelWrite

The excelWrite contains several methods to perform various operations on an Excel file. The class uses the Apache POI library, which is a popular Java library for working with Microsoft Office documents. The excelWrite class contains four methods: deleteCustomer(),createNewCustomer(), addCurrentDateToColumn() and addCurrReadingToColumn().

The deleteCustomer() method takes two arguments: a filePath string that specifies the path to the Excel file to be modified, and an integer customerId that represents the ID of the customer to be deleted. The method first reads in the Excel file using a FileInputStream object and creates an XSSFWorkbook object to represent the workbook. The method then gets the first sheet in the workbook using the getSheetAt() method.

Next, the method creates a FormulaEvaluator object using the getCreationHelper() method of the workbook object. This object is used to evaluate any formulas in the Excel sheet. The method then initializes a variable named rowId to -1, which will be used to keep track of the row containing the customer ID to be deleted.

The method then loops over each row in the sheet using a for loop that iterates over a Row object. Within the loop, the method gets the first cell of the current row using the getCell() method and assigns it to a Cell object named cell. The method checks if the cell is a numeric cell and if its value matches the customerId parameter using the getCellType() and getNumericCellValue() methods. If there is a match, the method assigns the row number to the rowId variable and exits the loop using the break statement.

If rowId is still -1 after the loop, it means that no row was found with the given customer ID, and the method prints a message to the console and returns without modifying the Excel file. Otherwise, the method gets the row to be deleted using the getRow() method and removes it from the sheet using the removeRow() method. The method then writes the modified workbook to the Excel file using a FileOutputStream object and the write() method of the XSSFWorkbook object. Finally, the method closes the output stream and prints a message to the console indicating that the customer has been successfully deleted.

The createNewCustomer() method takes a single argument: a filePath string that specifies the path to the Excel file to be modified. The method uses a Scanner object to prompt the user for the name and phone number of the new customer. It then reads in the Excel file using a FileInputStream object and creates an XSSFWorkbook object to represent the workbook. The method gets the first sheet in the workbook using the getSheetAt() method and gets the last row of the sheet using the getLastRowNum() method.

The method then gets the last cell of the last row using the getRow() method to get the row and the getCell() method to get the cell. The cell contains the ID of the last customer, so the method increments the ID by 1 and assigns it to a variable named newCustomerId. The method then creates a new row using the createRow() method and assigns it to a Row object named newRow. The method creates a cell for each column of the row using the createCell() method and assigns it to a Cell object.

The method sets the value of the cell in the first column to newCustomerId, sets the value of the cell in the second column to the customer name, and sets the value of the cell in the third column to the customer's email address. Finally, the method saves the changes to the workbook using the write() method.

After the changes are saved, the method returns the newly generated customer ID so that it can be used for future operations. This completes the process of adding a new customer to the workbook.

Overall, this method is useful for adding new data to an existing Excel workbook programmatically. It demonstrates the use of various methods in the Apache POI library for creating, updating, and saving Excel workbooks.

The Transactions class is a Java class that inherits from the excelWrite class. It contains a public static method generateBill that generates a bill for a given customer ID.

The method takes in two parameters, filePath and customerId, which are the path to the Excel file containing customer information and the customer ID for whom the bill needs to be generated, respectively. The method throws an IOException if there is an error in reading the file.

The first part of the method reads the Excel file using FileInputStream and XSSFWorkbook. It then gets the first sheet from the workbook using getSheetAt and creates a FormulaEvaluator to evaluate formulas in the sheet.

The method then loops through each row in the sheet to find the row with the given customer ID. It does this by checking the value in the first cell of each row against the given customer ID. If a row is found, its row number is stored in rowId. If no row is found, the method prints an error message and returns.

If a row is found, the method extracts relevant information from the row, such as the previous reading, current reading, due date, and payment status. It then calculates the units consumed and the amount to be paid based on the difference between the previous and current readings and a fixed rate of 5.0 per unit.

The method then prints the bill information, including customer ID, name, phone number, previous reading, current reading, units consumed, amount to be paid, and due date. It also prompts the user to confirm whether they want to pay the bill now.

If the user confirms payment, the method calls a private method payBill to update the payment status in the Excel sheet. It then writes the updated sheet to the Excel file using FileOutputStream and closes the output stream.

In summary, the Transactions class is a utility class that provides a method for generating bills for a given customer ID from an Excel sheet. It reads and writes Excel files using FileInputStream, FileOutputStream, XSSFWorkbook, and FormulaEvaluator. It also calculates the amount to be paid and updates the payment status in the Excel sheet.

This is a Java class called "App" that contains the main method which serves as the entry point of the program. The purpose of this program is to simulate a system for managing customer data and generating bills based on readings.

The class has several methods and variables that are used throughout the program. Let's take a closer look at them:

clearScreen() - This method is used to clear the console screen.

main(String args[]) - This is the main method of the class that takes an array of String arguments as a parameter. It starts by declaring and initializing several variables such as excelRead, excelWrite, and Transactions.

excelRead and excelWrite are objects of classes that handle reading and writing data to and from Excel files respectively. Transactions is a class that contains methods for generating bills based on customer readings.

customerID is a double variable that is used to store the ID of the customer whose data is being accessed.

Scanner in is an object of the Scanner class that is used to read input from the console.

userName and passWord are String variables that are used to store the username and password entered by the user.

filePath is a String variable that stores the path of the Excel file that is being used to store the customer data.

Admin is a boolean variable that is used to determine whether the user has administrative privileges.

The main method starts by prompting the user for their username and password. If the entered credentials match the hardcoded values, the Admin variable is set to true. If the credentials are invalid, an exception is thrown and caught by the catch block which displays an error message to the user.

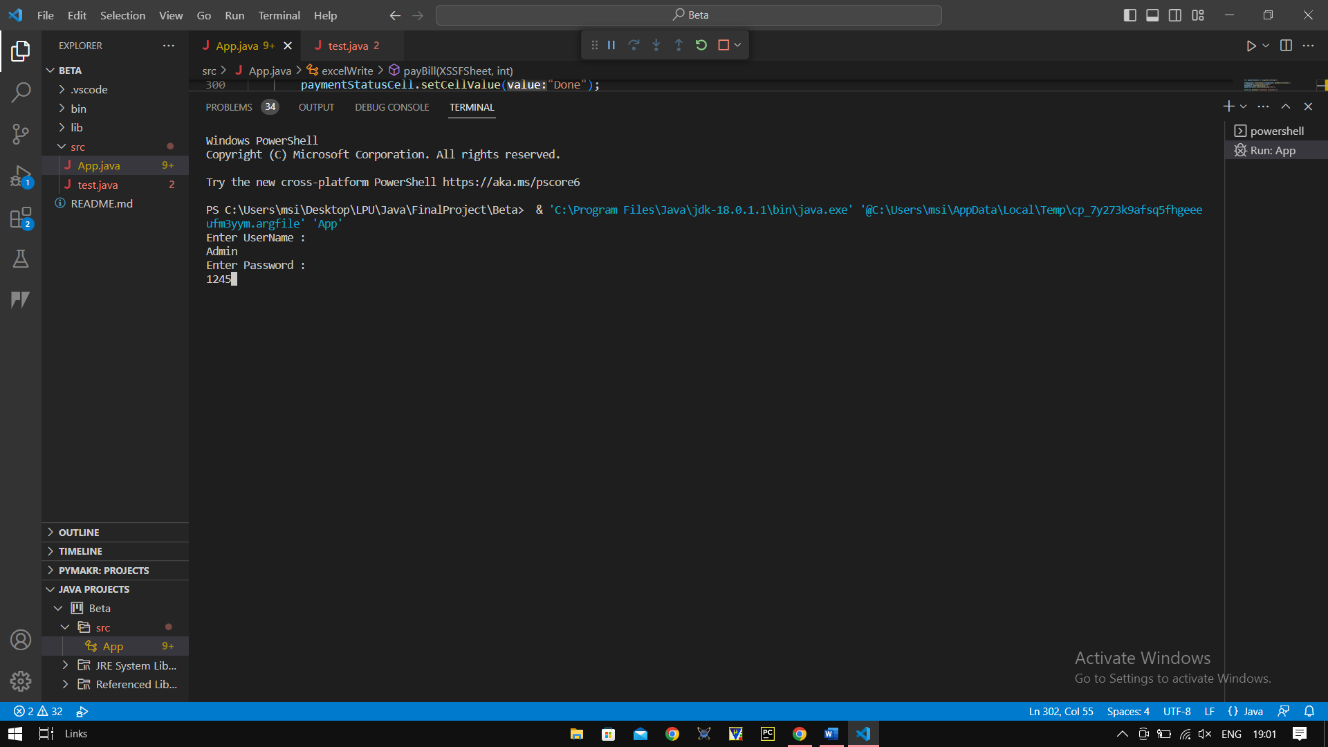
If the user has administrative privileges, a menu is displayed that allows the user to perform various operations such as showing all data, adding new customers, adding new readings, generating bills, and deleting customers.

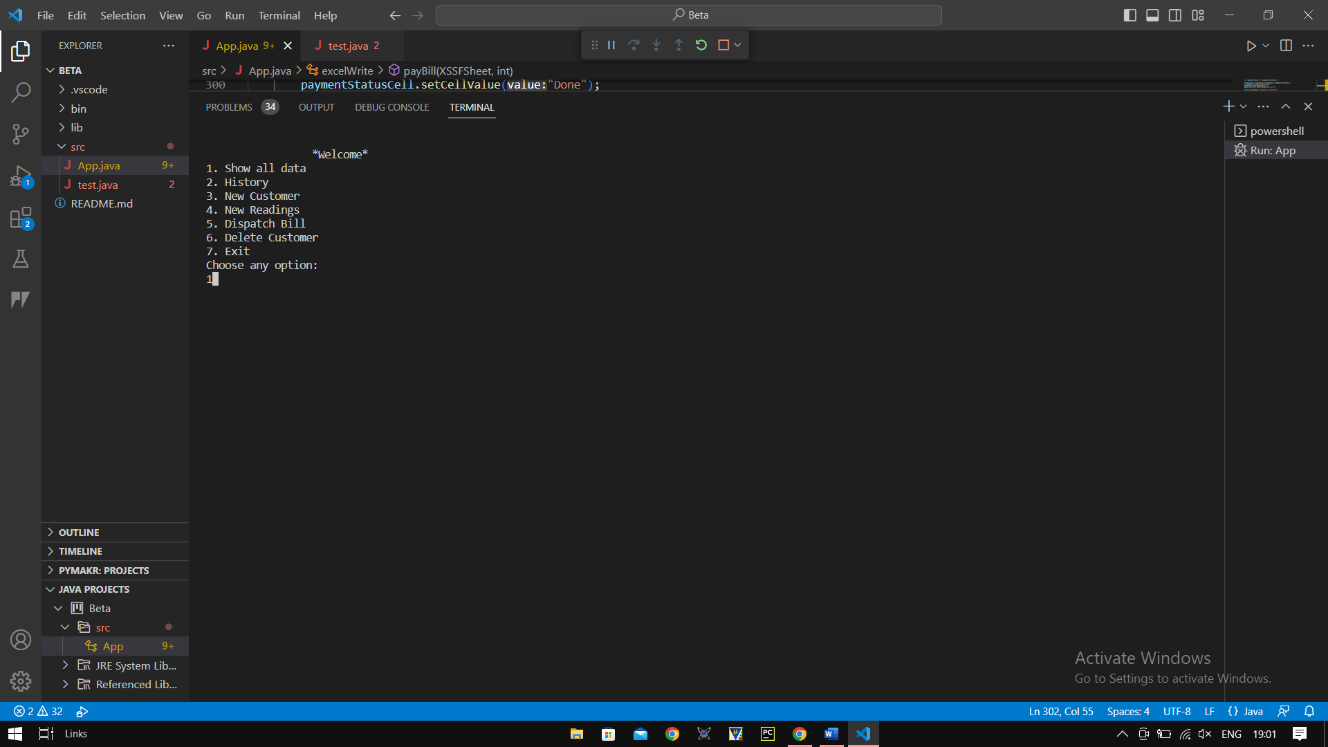
The user's choice is read using the Scanner object in and a switch statement is used to execute the appropriate method based on the user's choice.

After executing the chosen operation, the user is given the option to go back to the main menu or exit the program. If the user chooses to go back to the main menu, the menu is displayed again. If the user chooses to exit the program, the Admin variable is set to false and the program terminates.

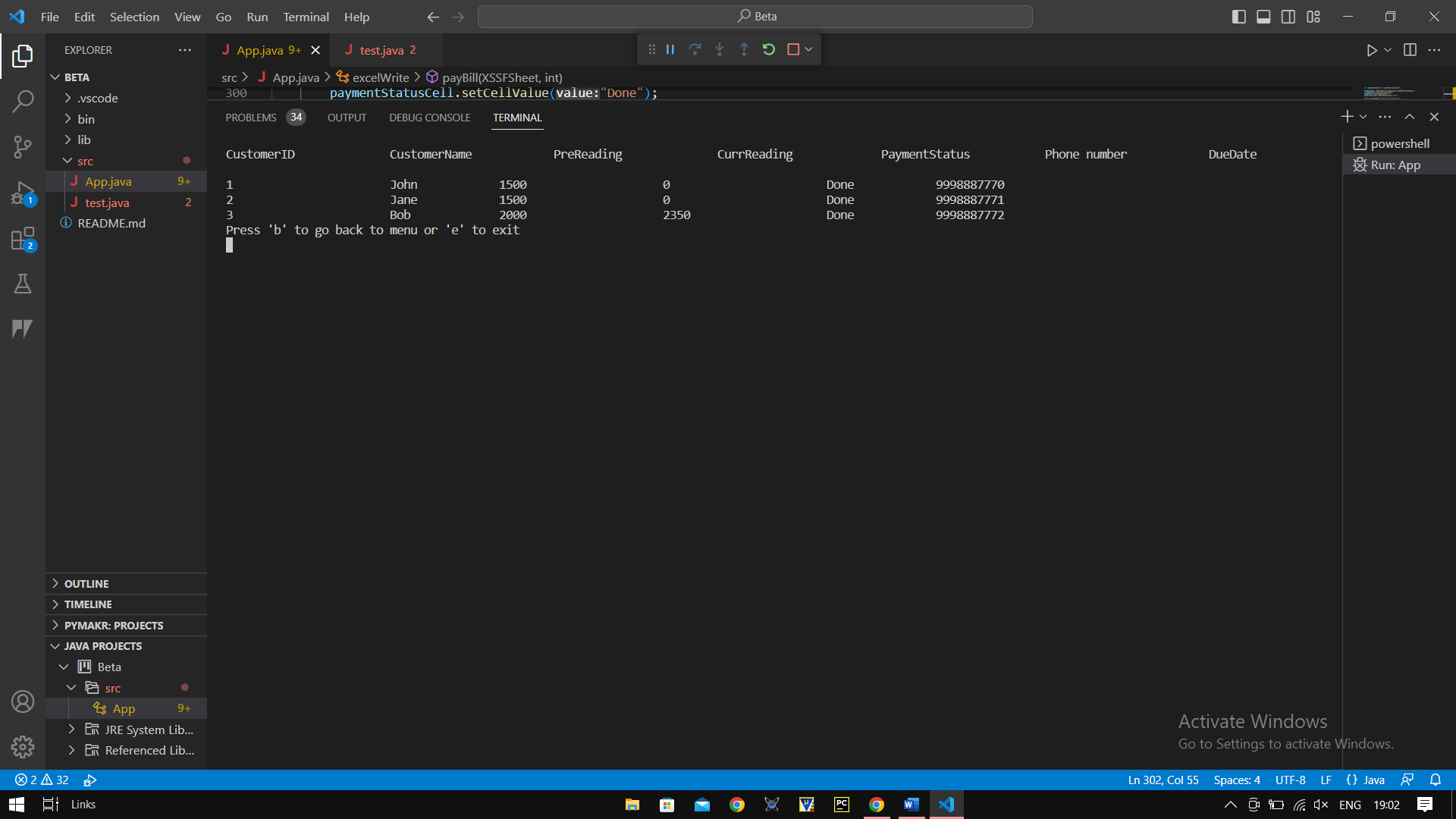
Overall, this program provides a simple interface for managing customer data and generating bills based on customer readings using an Excel file as a data store.

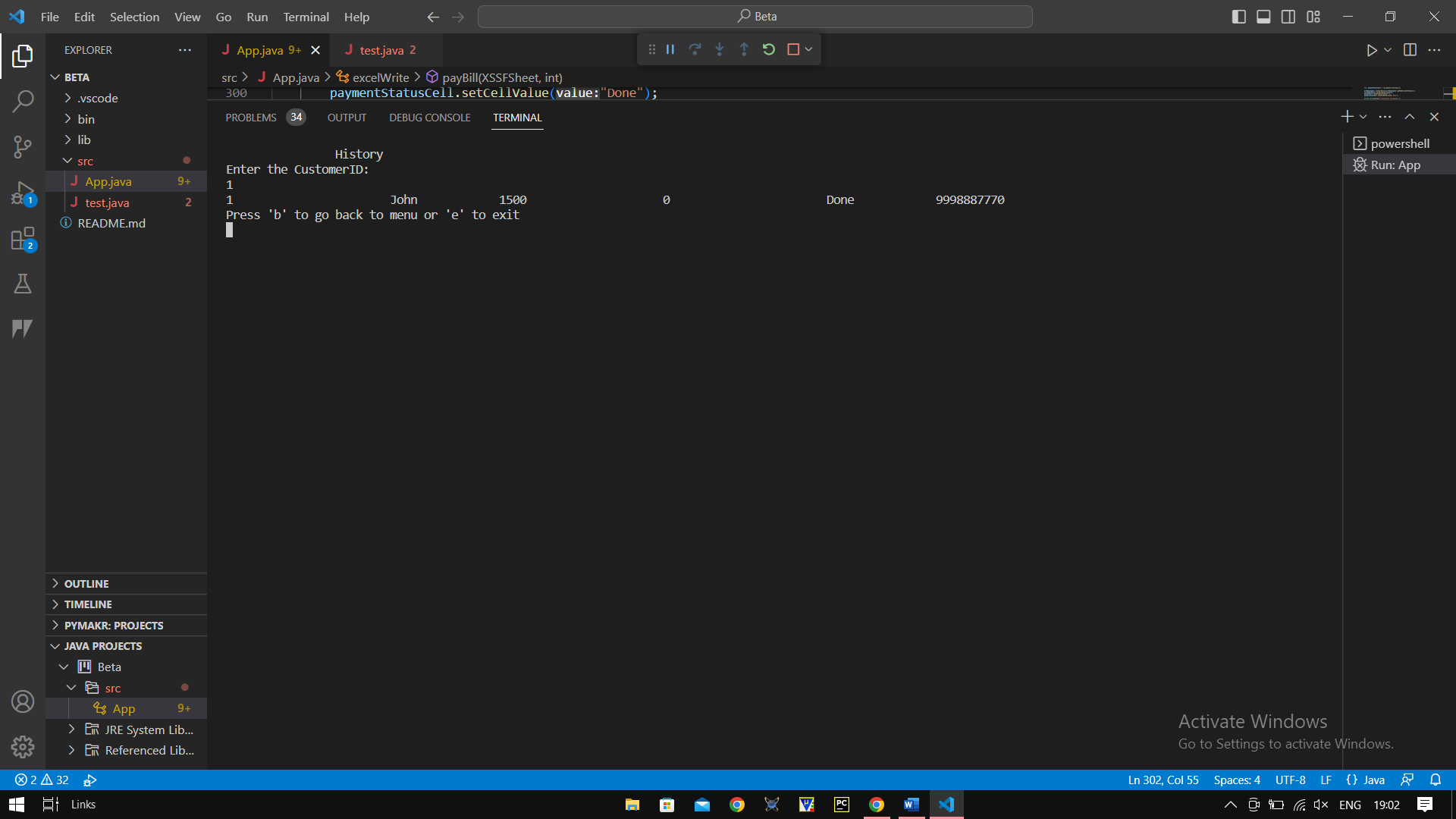
IV. User Interface

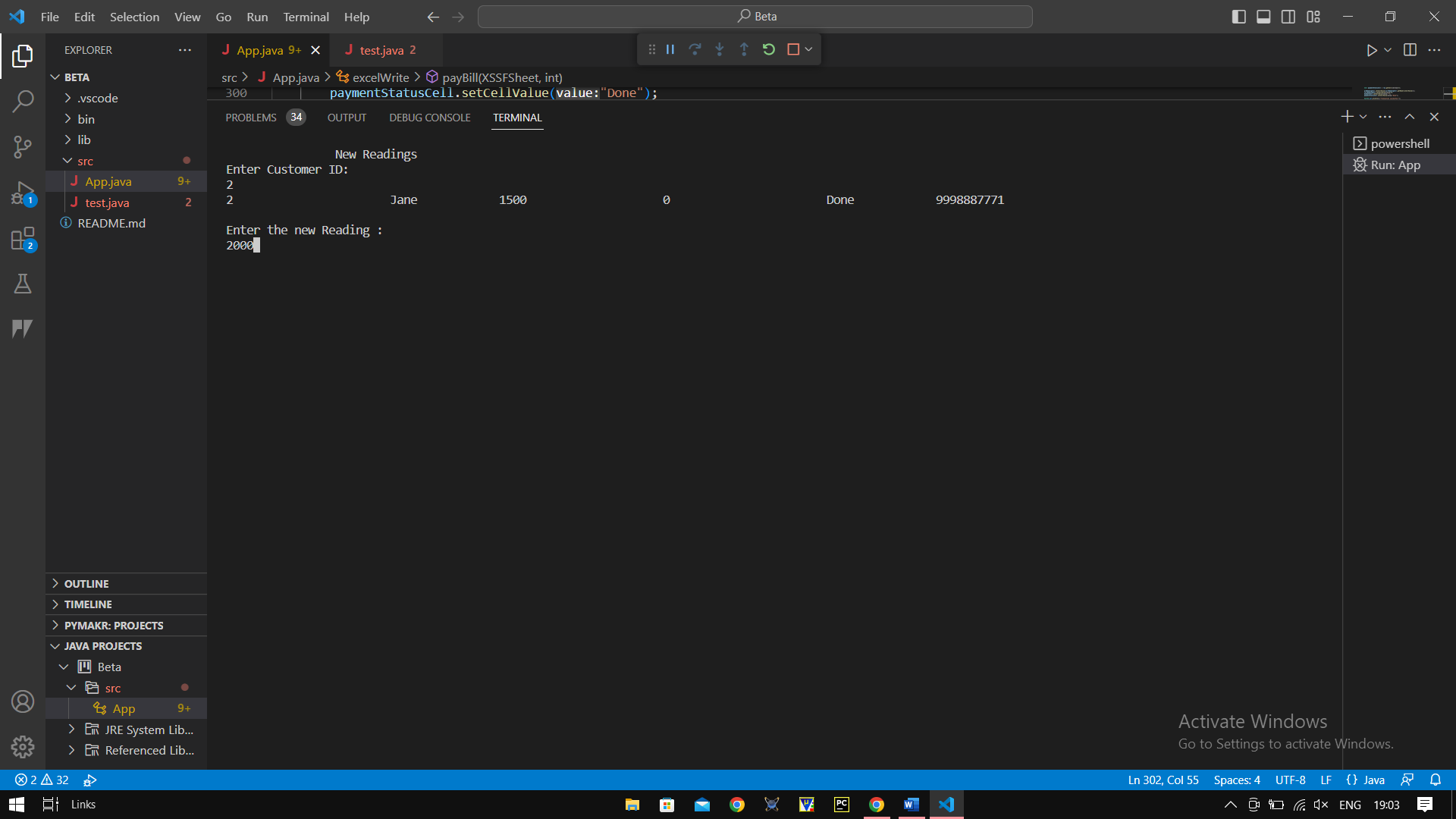


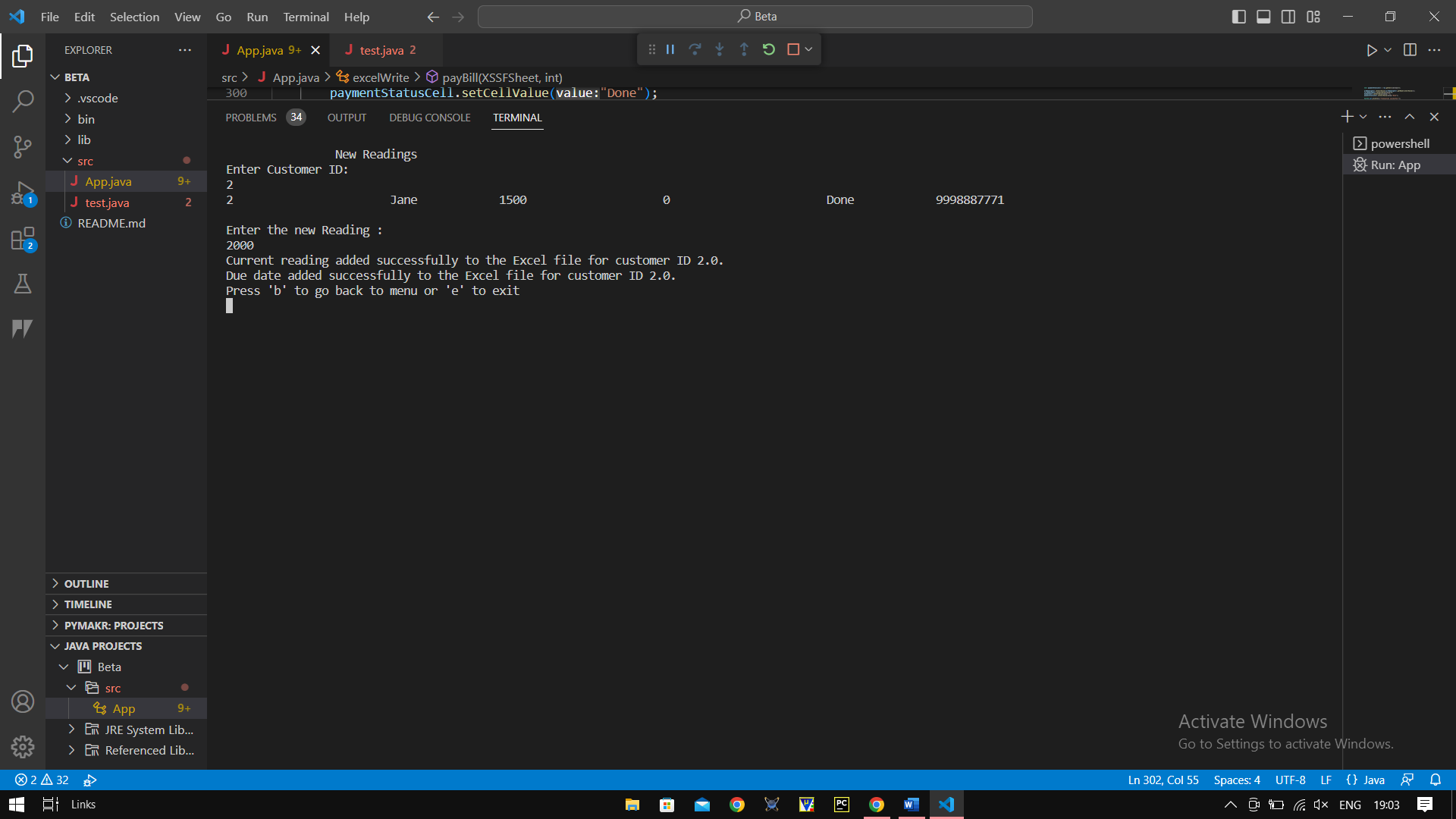


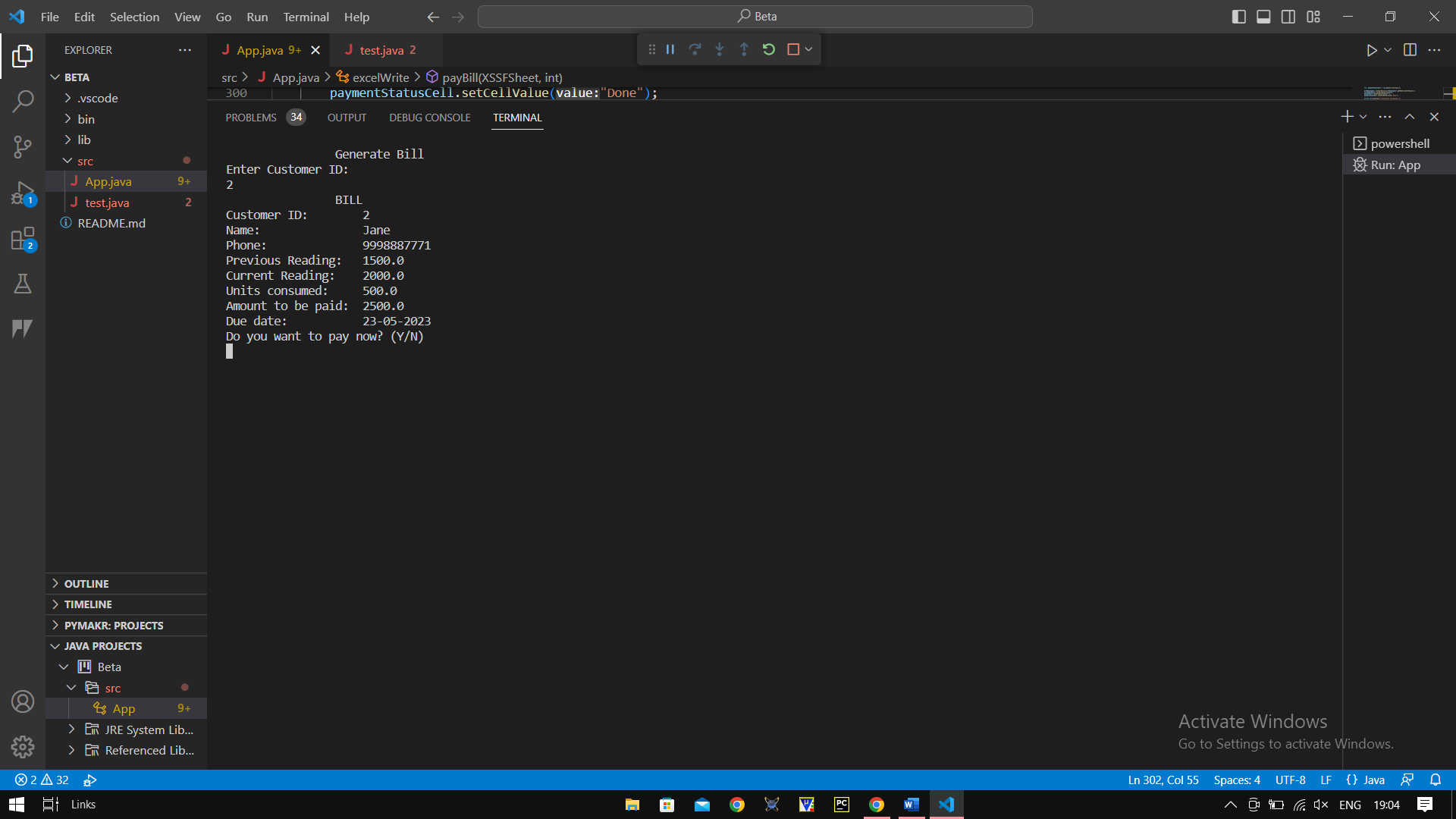
V. Testing

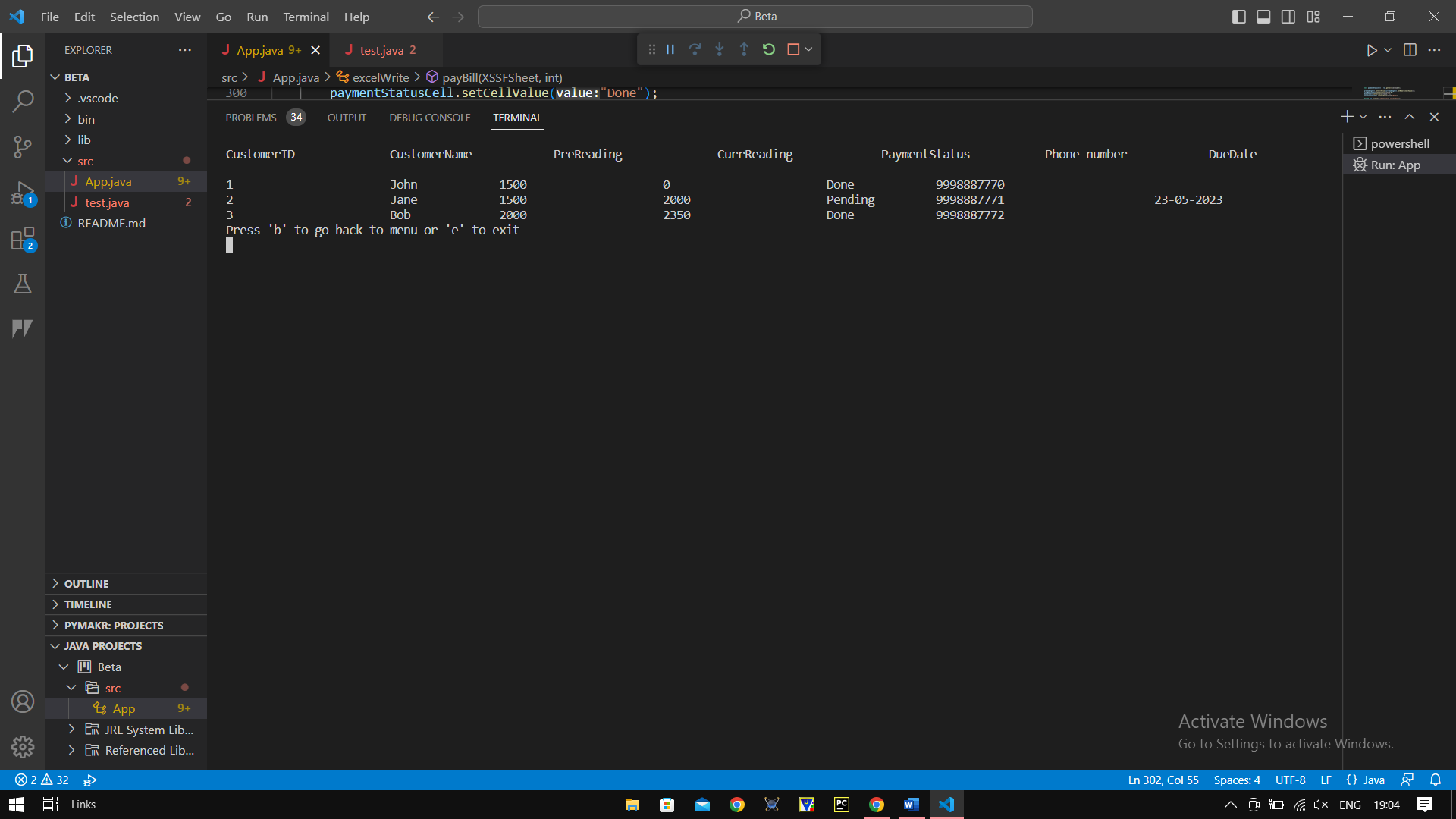


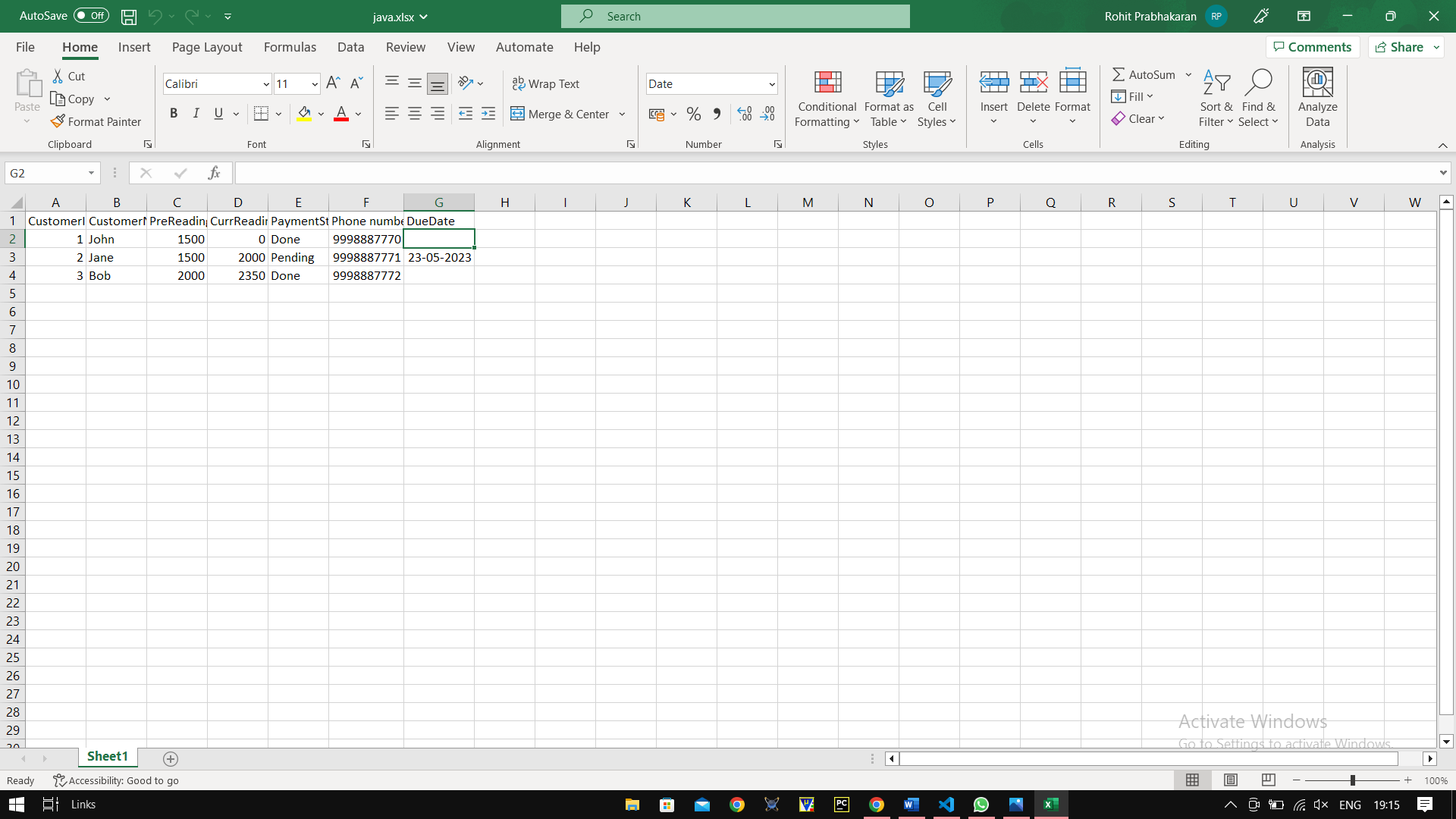












VI. Conclusion

In conclusion, the Electricity Bill Management System presented in this report is a practical solution for managing customer information, readings, and billing for electricity service providers. The system uses Java programming language and Apache POI library for reading, writing, and manipulating Excel files. The system is also scalable, allowing for the addition of new features and functionalities as required.

The use of Apache POI library provides flexibility in the system's functionalities, including the ability to read and write Excel files, format cells, calculate formulas, and retrieve data from specific cells. The system's user interface is intuitive and easy to use, with features such as creating new customers, adding readings, generating bills, and deleting customers. The system has demonstrated its ability to handle large amounts of customer data and transactions with ease. The use of Java programming language has also made the system secure and robust, with the ability to handle any errors that may arise.

Overall, the Electricity Bill Management System provides a reliable and efficient solution for electricity service providers to manage their billing and customer information. By utilizing the Apache POI library, the system can handle large amounts of data while providing a user-friendly interface. This system can be further enhanced by adding more features such as graphical representation of customer data and integrating with a database for better data management. It is a great example of how technology can be used to simplify complex processes and make them more efficient. With further development, the system can be improved to include more advanced features and capabilities, making it an even more powerful tool for managing electricity accounts.