Java Program: Main Menu and Functionality Overview

This document provides an overview of a Java program that implements a main menu system for managing various business operations. The program includes functionalities for project management, supply chain inventory, financial management, human resources, and report generation. It also includes an authentication mechanism to ensure secure access.

Program Structure

Main Method

The `main` method is the entry point of the program. It handles user authentication and presents a menu for selecting different management functionalities.

```
public static void main(String[] args) throws Exception {    try (Scanner scanner = new Scanner(System.in)) {        if (!authenticate(scanner)) {            System.out.println("Authentication failed. Program terminating.");            return;        }        int choice = 0;        do {            displayMenu();            String input = scanner.nextLine();            if (input.isEmpty()) {                continue;            }            choice = Integer.parseInt(input);            switch (choice) {                case 1 -> manageProject(scanner);                case 2 -> manageSupplyChainInventory(scanner);                case 3 -> manageFinancialManagement(scanner);                case 4 -> manageHumanResources(scanner);                case 5 -> generateReport();                case 6 -> {                    System.out.println("Exiting program...");                    System.exit(0);                }                default -> System.out.println("Invalid choice! Please try again.");            }        } while (choice != 6);    }    displayMenu();}
```

Authentication

The program begins by authenticating the user. If authentication fails, the program terminates with a message.

Menu Display and User Input

The program displays a menu of options and waits for the user to input a choice. The menu options are:

1. Manage Project
2. Manage Supply Chain Inventory
3. Manage Financial Management
4. Manage Human Resources
5. Generate Report
6. Exit Program

Functionality Options

- **Manage Project**: Invokes the `manageProject(scanner)` method to handle project-related tasks.
- **Manage Supply Chain Inventory**: Calls the `manageSupplyChainInventory(scanner)` method for inventory management.
- **Manage Financial Management**: Utilizes the `manageFinancialManagement(scanner)` method to oversee financial operations.
- **Manage Human Resources**: Engages the `manageHumanResources(scanner)` method for HR tasks.
- **Generate Report**: Executes the `generateReport()` method to produce reports.
- **Exit Program**: Exits the program gracefully.

Error Handling

The program includes basic error handling for invalid menu choices, prompting the user to try again.

Conclusion

This Java program provides a structured approach to managing various business operations through a simple command-line interface. It ensures secure access through authentication and offers a range of functionalities to handle different aspects of business management.

Java Authentication Method Documentation

This document provides an overview and explanation of the `authenticate` method implemented in Java. The method is designed to authenticate a user by verifying their username and password against a predefined set of correct credentials.

Method Overview

The `authenticate` method is a static method that takes a `Scanner` object as an argument. It is used to prompt the user for their username and password, and it attempts to authenticate the user with a maximum of three attempts.

Method Signature

`private static boolean authenticate(Scanner scanner)`

Parameters

- `scanner`: An instance of `Scanner` used to read user input from the console.

Return Value

- Returns `true` if the user is successfully authenticated.
- Returns `false` if the user fails to authenticate after three attempts.

Authentication Process

1. **Initialization**: The method initializes a variable `attempts` to 3, representing the maximum number of authentication attempts allowed.
2. **User Prompt**: The method enters a loop where it prompts the user to enter their username and password.
3. **Credential Verification**:
   - The method checks if the entered username and password match any pair in the `CORRECT_CREDENTIALS` array using the `Arrays.stream().anyMatch()` method.
   - If a match is found, the method prints "Login successful!" and returns `true`.
4. **Invalid Credentials Handling**:
   - If the credentials do not match, the method decrements the `attempts` counter.
   - It then informs the user of the invalid credentials and displays the number of remaining attempts.
5. **Failure Notification**:
   - If the user exhausts all attempts without successful authentication, the method prints "Login failed. No more attempts remaining." and returns `false`.

Example Usage

Below is an example of how the `authenticate` method might be used within a Java application:

```
import java.util.Scanner;public class AuthenticationExample {
private static final String[][] CORRECT_CREDENTIALS = {
{"user1", "password1"},        {"user2", "password2"}    };
public static void main(String[] args) {        Scanner scanner =
new Scanner(System.in);        boolean isAuthenticated =
authenticate(scanner);        if (isAuthenticated) {
System.out.println("Access granted.");        } else {
System.out.println("Access denied.");        }
scanner.close();    }    private static boolean
authenticate(Scanner scanner) {        // Method implementation as
described above    }}
```

Conclusion

The `authenticate` method provides a simple yet effective way to manage user authentication in a console-based Java application. By allowing multiple attempts and providing feedback on the number of remaining attempts, it enhances the user experience while maintaining security.

Project Management System Documentation

Overview

This document provides an overview of the data structures used in a project management system. The system is designed to manage information related to projects, budgets, and employees. The data is stored in arrays, with each array dedicated to a specific type of information.

Data Structures

Projects

- **Array Name:** `projects`
- **Type:** `String[][]`
- **Size:** 100 rows, 7 columns
- **Purpose:** Stores information about various projects.
- **Columns Description:**
  a. Project ID
  b. Project Name
  c. Start Date
  d. End Date
  e. Status
  f. Budget ID
  g. Manager ID
- **Current Count:** `projectCount`(tracks the number of projects stored)

Budget Data

- **Array Name:** `budgetData`
- **Type:** `String[][]`
- **Size:** 100 rows, 2 columns
- **Purpose:** Stores budget information related to projects.
- **Columns Description:**
  a. Budget ID
  b. Amount

Employees

- **Array Name:** `employees`
- **Type:** `String[][]`
- **Size:** 100 rows, 3 columns
- **Purpose:** Stores information about employees.
- **Columns Description:**
  a. Employee ID
  b. Employee Name
  c. Role
- **Current Count:** `employeeCount`(tracks the number of employees stored)

Usage

These arrays are used to manage and retrieve data efficiently within the system. The fixed size of the arrays implies a limitation on the number of entries that can be stored, which is currently set to 100 for each category. The system uses counters (`projectCount`and `employeeCount`) to keep track of the number of entries in the `projects`and `employees`arrays, respectively.

Considerations

- **Scalability:** The current implementation uses fixed-size arrays, which may need to be adjusted for larger datasets.
- **Data Integrity:** Ensure that the IDs used in the arrays are unique to maintain data integrity.
- **Performance:** Accessing data in arrays is efficient, but consider using more dynamic data structures if the dataset grows significantly.
- This documentation serves as a guide to understanding the basic structure and functionality of the project management system's data handling components.

Java Program: Main Menu and Functionality Overview

This document provides an overview of a Java program's main method, which includes user authentication, menu display, and various management functionalities. The program is designed to handle different aspects of a business management system, such as project management, supply chain inventory, financial management, human resources, and report generation.

Program Flow

1. **Authentication:**
   - The program begins by authenticating the user. If authentication fails, the program terminates with a message: "Authentication failed. Program terminating."
2. **Main Menu:**
   - Once authenticated, the user is presented with a menu of options. The menu is displayed in a loop until the user chooses to exit the program.
3. **User Choices:**
   - The user can select from the following options by entering the corresponding number:
     - **1:** Manage Project
     - **2:** Manage Supply Chain Inventory
     - **3:** Manage Financial Management
     - **4:** Manage Human Resources
     - **5:** Generate Report
     - **6:** Exit Program
4. **Handling User Input:**
   - The program reads user input using a `Scanner`. If the input is empty, it prompts the user again.
   - The input is parsed as an integer to determine the user's choice.
   - A `switch` statement is used to execute the corresponding method based on the user's choice.
5. **Functionality:**
   - **Manage Project:** Calls `manageProject(scanner)` to handle project-related tasks.
   - **Manage Supply Chain Inventory:** Calls `manageSupplyChainInventory(scanner)` for inventory management.
   - **Manage Financial Management:** Calls `manageFinancialManagement(scanner)` for financial operations.
   - **Manage Human Resources:** Calls `manageHumanResources(scanner)` for HR tasks.
   - **Generate Report:** Calls `generateReport()` to produce a report.
   - **Exit Program:** Outputs "Exiting program..." and terminates the program using `System.exit(0)`.
6. **Invalid Input Handling:**
   - If the user enters an invalid choice, the program outputs "Invalid choice! Please try again."

Conclusion

This Java program provides a structured approach to managing various business operations through a user-friendly menu interface. It ensures secure access through authentication and handles user input efficiently to perform different management tasks.

Project Management System

This document outlines the functionality of a simple project management system implemented in Java. The system allows users to manage projects through a console-based interface. Below is a detailed explanation of the `manageProject` method, which serves as the main menu for project management operations.

Method: `manageProject`

The `manageProject` method is responsible for presenting the user with a menu of options related to project management. It uses a `Scanner` object to read user input and performs actions based on the user's choice.

Menu Options

1. **Add Project**: Allows the user to add a new project to the system.
2. **List Projects**: Displays a list of all existing projects.
3. **Update Project Status**: Enables the user to update the status of an existing project.
4. **Return to Main Menu**: Exits the project management menu and returns to the main menu of the application.

Implementation Details

- **Input Handling**: The method prompts the user to enter a choice between 0 and 3. It reads the input using `scanner.nextInt()` and then clears the buffer with `scanner.nextLine()`.
- **Switch Statement**: A switch statement is used to handle the user's choice:
  - **Case 1**: Calls `addProject(scanner)` to add a new project and then recursively calls `manageProject(scanner)` to return to the menu.
  - **Case 2**: Calls `listProjects(scanner)` to list all projects and then recursively calls `manageProject(scanner)` to return to the menu.
  - **Case 3**: Calls `updateProjectStatus(scanner)` to update a project's status and then recursively calls `manageProject(scanner)` to return to the menu.
  - **Case 0**: Exits the method, returning control to the main menu.
  - **Default**: Handles invalid input by displaying an error message and recursively calling `manageProject(scanner)` to prompt the user again.

Error Handling

If the user enters an invalid choice, the system displays an error message and prompts the user to try again. This ensures that the user remains in the project management menu until a valid option is selected or they choose to return to the main menu.

Recursion

The method uses recursion to return to the project management menu after completing an operation or encountering an invalid input. This approach simplifies the flow of control and ensures that the user can continuously interact with the menu without restarting the application.

Conclusion

The `manageProject` method provides a straightforward and user-friendly interface for managing projects. By leveraging a switch statement and recursion, the method efficiently handles user input and guides the user through various project management tasks. This design ensures that users can easily add, list, and update projects within the system.

Project Management System: Add Project Method

This document provides a detailed explanation of the `addProject` method within the project management system. This method is responsible for adding new projects to the system, capturing essential project details, and calculating the project cost with VAT.

Method: addProject

The `addProject` method is a static method that facilitates the addition of a new project to the system. It interacts with the user through a console-based interface to gather necessary project information and compute the total project cost.

Functionality

1. **User Input**: The method prompts the user to enter various details about the project, including:
    - **Project Name**: Expected format is `OWNER_(PPA,EPC)_kWp`.
    - **Project Start Date**: In the format `YYYY-MM-DD`.
    - **Project End Date**: In the format `YYYY-MM-DD`.
    - **Project ID**: A unique identifier in the format `0x`.
    - **Project Size**: The size of the project in kilowatt peak (kWp).
2. **Cost Calculation**:
    - **Cost Per Watt**: Assumed to be 5 Thai baht.
    - **Project Cost**: Calculated by multiplying the project size (converted to watts) by the cost per watt.
    - **Adjustment**: The initial project cost is increased by 20%.
    - **VAT**: A 7% VAT is added to the adjusted project cost.
    - The final project cost, including VAT, is stored in the project details.
3. **Project Storage**: The project details, including the calculated cost, are stored in a two-dimensional array `projects`, with each project occupying a row.
4. **Confirmation**: Upon successful addition, the method outputs a confirmation message displaying the project name and ID.

Implementation Details

- **Scanner Usage**: The method uses a `Scanner` object to read user input from the console. It handles both `String` and `double` inputs, ensuring the buffer is cleared appropriately after reading a `double`.
- **Project Count**: The variable `projectCount` is used to track the number of projects and determine the next available row in the `projects` array for storing new project details.

Example

When a user adds a project, they will see prompts for each required detail. After entering all information, the system calculates the cost and confirms the addition with a message like:

```
** Project : OWNER_PPA_100kWp | Project ID: 0x1234 | , added successfully!
```

Conclusion

The `addProject` method is a crucial component of the project management system, enabling users to efficiently add new projects with detailed cost calculations. By capturing comprehensive project data and ensuring accurate cost computation, the method supports effective project management and planning.

Java Method Documentation: `listProjects`

Overview

The `listProjects` method is a static function designed to display a list of projects stored in a multi-dimensional array. It is part of a larger application that manages project records. This method is invoked to print out the details of each project, including its name, ID, start and end dates, cost, and status.

Method Signature

`private static void listProjects(Scanner scanner)`Parameters

- `scanner`: An instance of `Scanner` used for user input. It is passed to the method to potentially handle further user interactions after listing the projects.

Functionality

1. **Check for Projects**:
   - The method first checks if there are any projects to display by evaluating `projectCount`. If `projectCount` is zero, it prints a message indicating that no projects are found and calls `manageProject(scanner)` to handle further actions.
2. **Display Projects**:
   - If projects exist, it prints a header `++ Projects Record : ++` to indicate the start of the project list.
   - It iterates over the `projects` array, which is assumed to be a two-dimensional array where each row represents a project and columns represent project attributes.
   - For each project, it prints:
     - **Project Number**: Sequential number starting from 1.
     - **Project Name**: Retrieved from `projects[i][0]`.
     - **Project ID**: Retrieved from `projects[i][3]`.
     - **Start Date**: Retrieved from `projects[i][1]`.
     - **End Date**: Retrieved from `projects[i][2]`.
     - **Project Cost**: Retrieved from `projects[i][4]`, formatted with commas for thousands, and appended with "THB" to denote the currency.
     - **Status**: Retrieved from `projects[i][6]`.

Example Output

`++ Projects Record : ++No.1 Project Name: Project Alpha | Project ID: 001, Start Date: 2024-01-01, End Date: 2024-12-31, Project Cost: 1,000,000 THB, Status: ActiveNo.2 Project Name: Project Beta | Project ID: 002, Start Date: 2024-02-01, End Date: 2024-11-30, Project Cost: 750,000 THB, Status: Completed`Notes

- The method assumes that the `projects` array and `projectCount` are properly initialized and populated elsewhere in the application.
- The method uses `String.format` to ensure that the project cost is displayed with proper formatting for readability.
- The `manageProject(scanner)` method is called when no projects are found, suggesting that it might provide options to add or manage projects.
- This documentation provides a comprehensive understanding of the `listProjects` method, its purpose, and its implementation details within the context of a project management application.

Java Method Documentation:

`updateProjectStatus`

Overview

The `updateProjectStatus` method is a static function designed to update the status of a project within a project management system. It interacts with the user via the console to select a project and update its status.

Method Signature

`private static void updateProjectStatus(Scanner scanner)`Parameters

- `Scanner scanner`: A `Scanner` object used to read user input from the console.

Functionality

1. **Check for Available Projects**:
    - The method first checks if there are any projects available (`projectCount == 0`). If no projects are available, it informs the user and calls `manageProject(scanner)` to handle project management tasks.
2. **Display Projects**:
    - If projects are available, it lists all projects with their IDs and names, allowing the user to select a project by its ID.
3. **User Input for Project Selection**:
    - Prompts the user to enter the project ID of the project they wish to update.
4. **User Input for Status Update**:
    - Prompts the user to enter a new status for the selected project. The status could be values like 'Bidding', 'In Progress', or 'Completed'.
5. **Update Project Status**:
    - Iterates through the list of projects to find the project with the matching ID.
    - Updates the status of the project if a match is found.
6. **Feedback to User**:
    - If the project status is successfully updated, it confirms the update to the user.
    - If no matching project is found, it informs the user of the failure due to a mismatch or non-existent project ID.

Edge Cases and Considerations

- **Input Validation**: The method assumes valid integer input for project ID and valid string input for the status. Additional validation may be required for robustness.
- **Project ID Matching**: The method uses `Integer.parseInt` to convert the project ID from the project data array, which assumes that the ID is stored as a string.
- **Array Indexing**: The method assumes that the project data is stored in a 2D array `projects` and that the project ID is located at index 3 and the status at index 6.

Example Usage

`Scanner scanner = new Scanner(System.in);updateProjectStatus(scanner);`This method is typically called within a larger project management application where `projectCount` and `projects` are defined and managed globally. It provides a simple interface for updating project statuses through console input.