# **Bank Account Management Application Documentation**

## **Table of Contents**

1. Project Overview
2. Setup Instructions
3. Class Descriptions
   * BankAccount
   * AccountDetailsWindow
4. Method Descriptions
5. Programming Principles
6. Usage

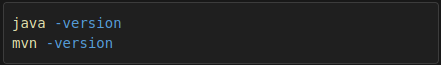
## **Project Overview**

The Simple Bank App is a Java-based application that provides a graphical user interface (GUI) for managing bank accounts. Users can create accounts, log in to existing accounts, and perform various banking operations such as deposits, withdrawals, and balance checks. The application uses JSON for data storage and SHA-256 hashing for password security.

## **Setup Instructions**

### **Prerequisites**

* Java 8 or later
* Maven (optional, for building if a pom.xml file is available)

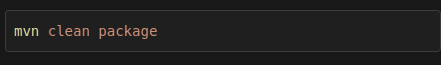


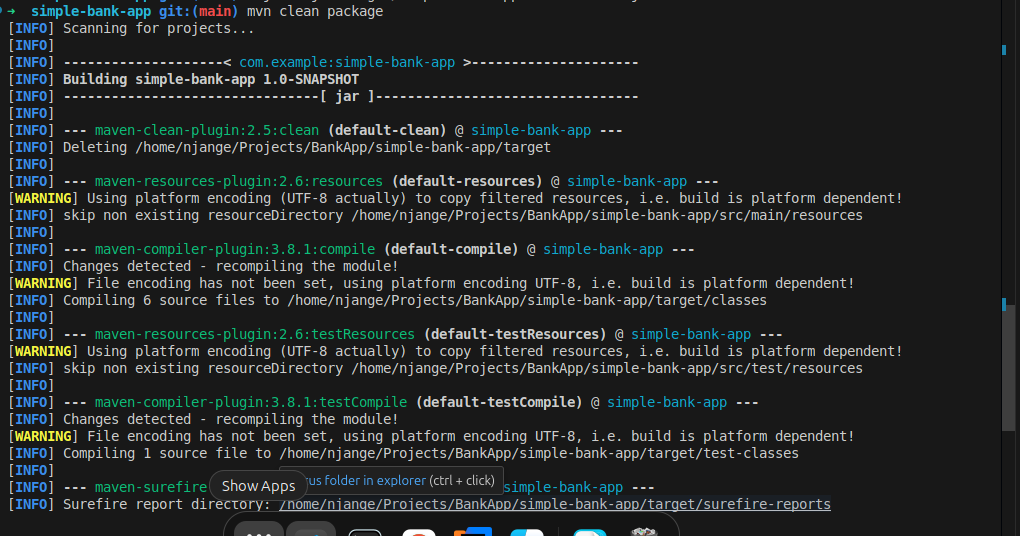
### **Running the Project**

1. **Extract the Codebase**Extract the downloaded or zipped folder containing the project files to a directory on your local machine.
2. **Navigate to the Project Directory**Open a terminal or command prompt, then navigate to the project folder.

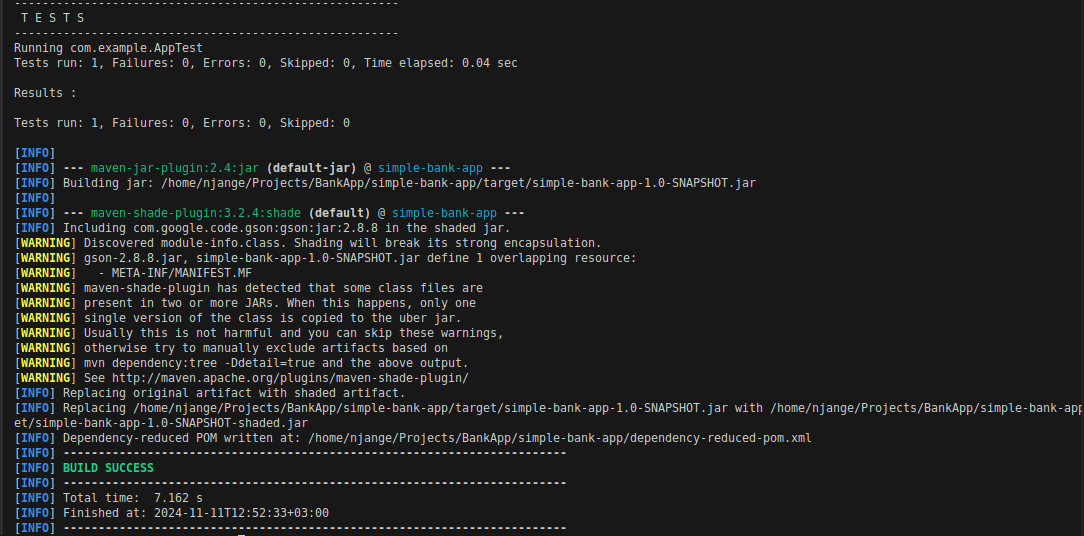
**Navigate to simple-bank-app**

1. **Compile the Code**Compile the Java files using:





**Tests done on Running**

****

1. **Run the Application**

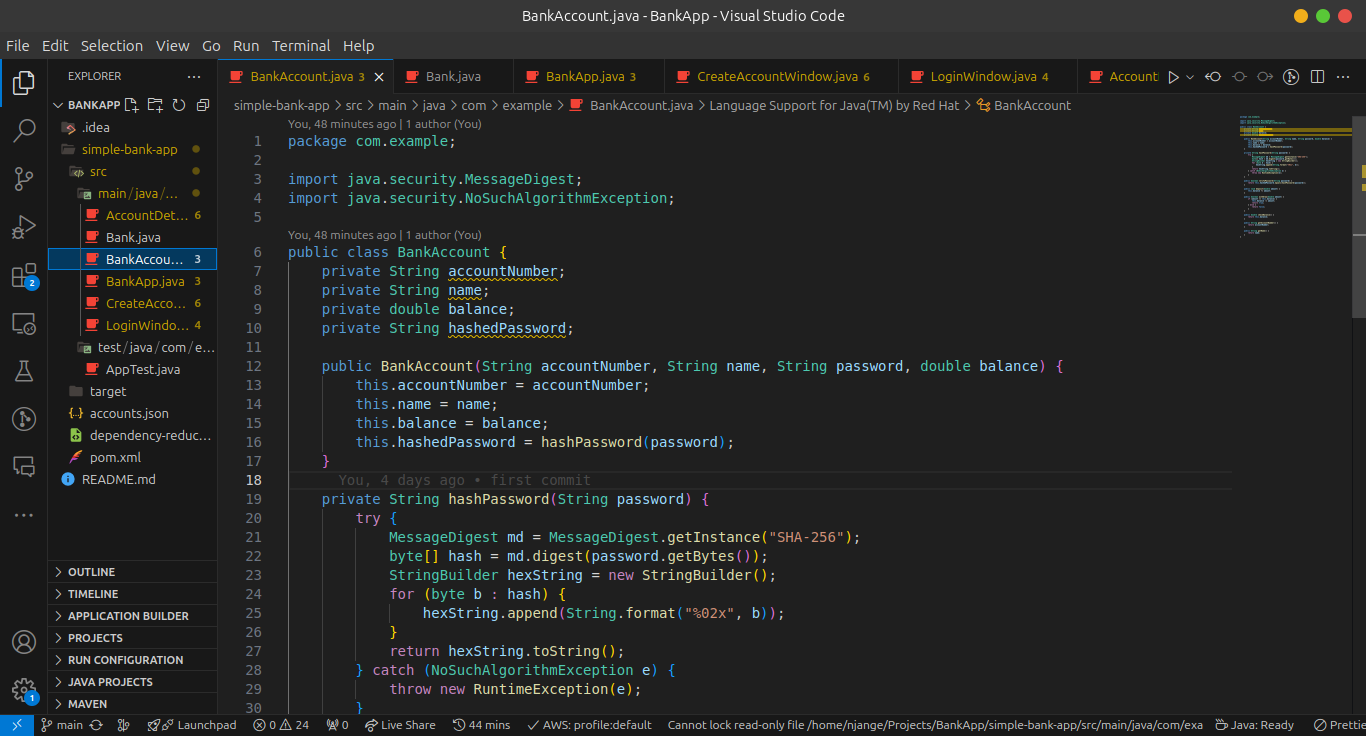
After compiling, you can launch the application with:



## **Class Descriptions**

### **BankAccount**

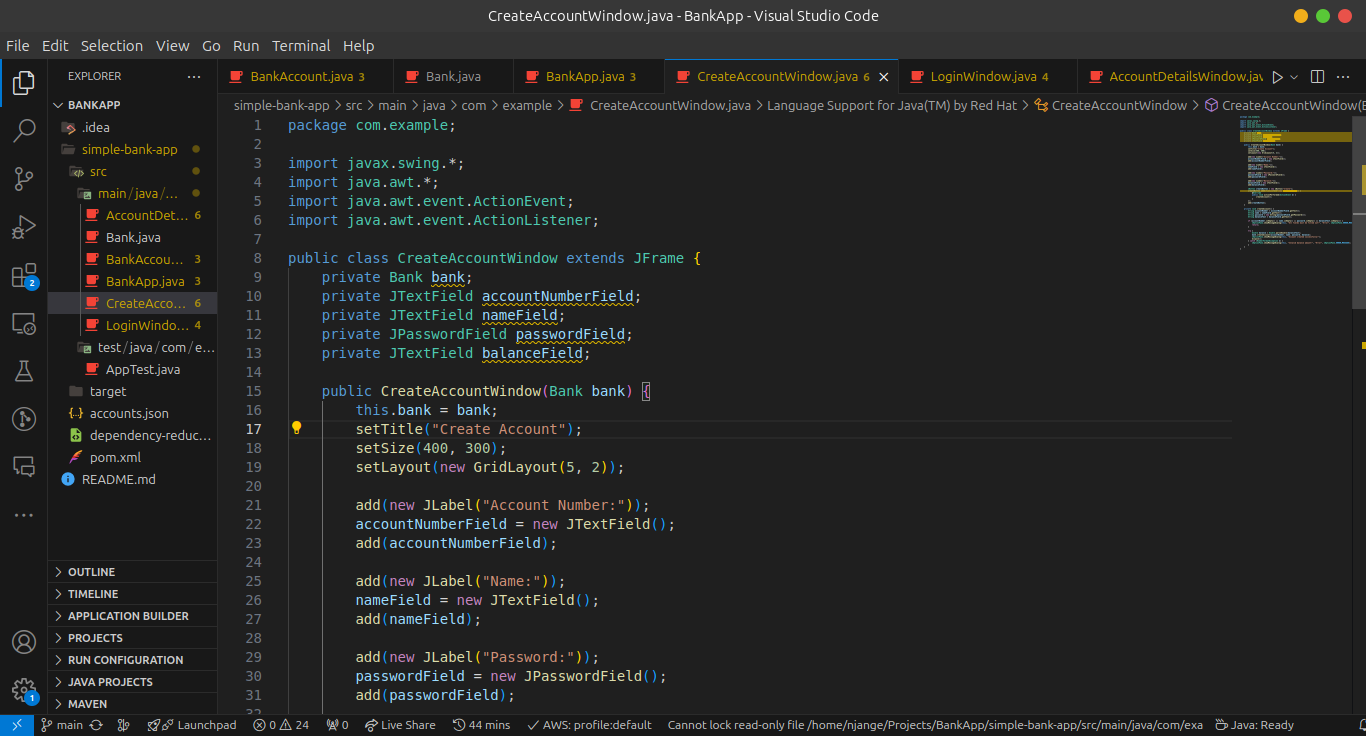
The BankAccount class represents an individual bank account and includes key methods for balance management.

* **Attributes:**
  + accountNumber: A unique identifier for each bank account.
  + balance: Stores the current account balance.
  + accountHolderName: The name associated with the account.
* **Methods:**
  + deposit(double amount): Adds funds to the account.
  + withdraw(double amount): Withdraws funds, provided there are sufficient funds.
  + getBalance(): Returns the account balance.
  + 
  + toString(): Outputs account details in a formatted string.

### **AccountDetailsWindow**

The AccountDetailsWindow class manages the graphical interface for viewing account details and performing transactions.

* **Attributes:**
  + bankAccount: An instance of BankAccount, representing the account being viewed.
* **Methods:**
  + initializeUI(): Sets up the graphical components for the window.
  + displayAccountInfo(): Displays details of the selected account.
  + processDeposit(): Handles deposit transactions based on user input.
  + processWithdrawal(): Handles withdrawal transactions based on user input.



## **Method Descriptions**

### **BankAccount Methods**

* **deposit(double amount)**: Adds the specified amount to the current balance.
* **withdraw(double amount)**: Deducts amount from the balance if enough funds are available.
* **getBalance()**: Returns the current balance of the bank account.
* **toString()**: Formats and returns account details as a string.

### **AccountDetailsWindow Methods**

* **initializeUI()**: Sets up the graphical interface for the account details window.
* **displayAccountInfo()**: Refreshes and displays the account’s information on the GUI.
* **processDeposit()**: Prompts the user for a deposit amount and updates the balance.
* **processWithdrawal()**: Prompts for a withdrawal amount and updates the balance if sufficient funds are available.

## **Data Storage**

### **Overview**

In this Bank Account Management Application, data storage is primarily file-based, and account information is stored in a structured JSON format. This approach enables the application to retain data across sessions, ensuring users’ account details are not lost when the application closes.

### **Data Storage Format**

The application uses JSON (JavaScript Object Notation) to store account data. JSON is lightweight, human-readable, and easily parsed, making it ideal for storing structured data in key-value pairs.

## **Programming Principles**

### **Encapsulation**

Encapsulation is used to restrict direct access to the class attributes. For example, the BankAccount class hides the balance attribute by making it private and exposes methods such as deposit() and withdraw() to safely modify the balance.

### **Abstraction**

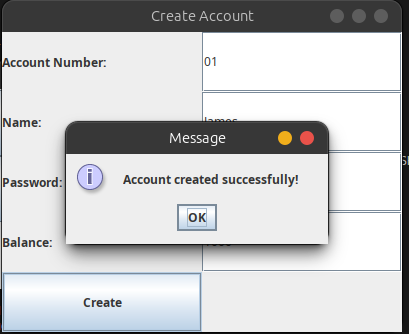
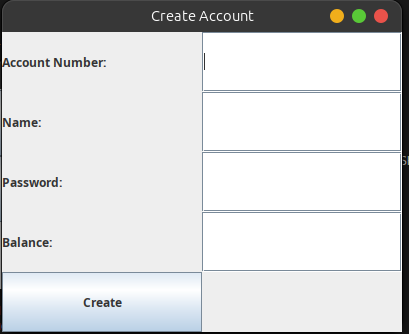
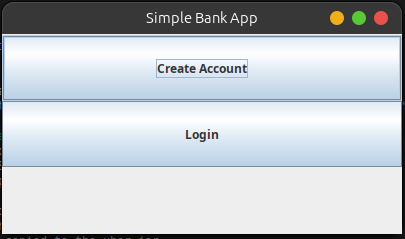
The BankAccount and AccountDetailsWindow classes are abstract representations of real-world entities, hiding the implementation details and only exposing necessary operations like deposit and withdraw, or account info display.

### **Separation of Concerns**

This project divides the responsibilities between the GUI (AccountDetailsWindow) and the business logic (BankAccount). This ensures a cleaner, more maintainable codebase.

## **Usage**

1. **Creating an Account**: Use BankAccount to instantiate a new account, providing the necessary details like account number, balance, and holder name.



1. **Launching the GUI**: Start the AccountDetailsWindow with a BankAccount instance to view and manage the account’s transactions.
2. **Performing Transactions**:
   * **Deposit**: Click the "Deposit" button in the GUI to add funds to the account.
   * **Withdraw**: Click "Withdraw" to remove funds from the account, given there’s a sufficient balance.

