Project Summary – Simply Banking

The "SimplyBanking" project is a **command-line Java application** that simulates basic banking operations. It allows users to:

- Create new bank accounts.
- Sign in to existing accounts.
- Check their account balance.
- Deposit funds.
- Withdraw funds.

It's built using Java and Maven, with JUnit 5 for unit testing.

How does it work?

The project is structured into a few key components that work together:

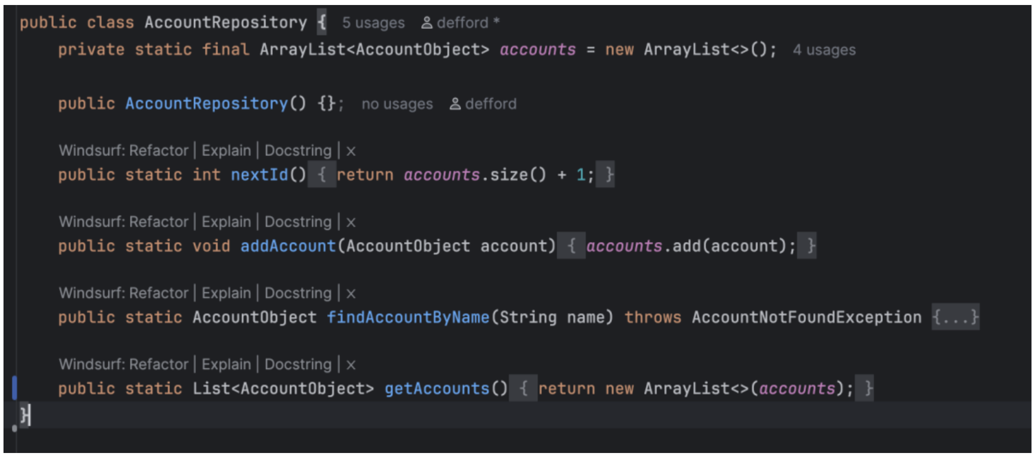
1. App.java **(User Interface):** This is what you run. It provides a text-based menu in your console. You interact with it by typing numbers to choose options (like "Create Account" or "Sign In") and entering information like your name, PIN, and amounts.
2. AccountObject.java **(Data Model):** This class is like a blueprint for a bank account. It defines what information an account holds: an ID, the account holder's name, a PIN for security, and the current balance.
3. AccountService.java **(Business Logic):** This class contains the "brains" for the banking operations. When you, for example, try to withdraw money, the App.java tells the AccountService.java what you want to do. The AccountService then checks if you have enough funds before allowing the withdrawal. It also handles creating new accounts and ensuring they get a unique ID.
4. AccountRepository.java **(Data Storage):** This class is responsible for keeping track of all the created bank accounts. It stores the accounts in a list in the computer's memory. It provides functions to add new accounts and find existing ones by name.
5. **Exception Handling:** The application uses custom error types (like AccountNotFoundException if you try to log into an account that doesn't exist, or InsufficientBalanceException if you try to withdraw too much) to manage problems smoothly.

In a nutshell: You interact with
App.java, which uses AccountService.java to perform actions. AccountService.java uses AccountObject.java to represent account data and AccountRepository.java to store and retrieve that data from memory.

Examples of Clean Code Principles

Example #1: Repository Class (Single Responsibility Principle)

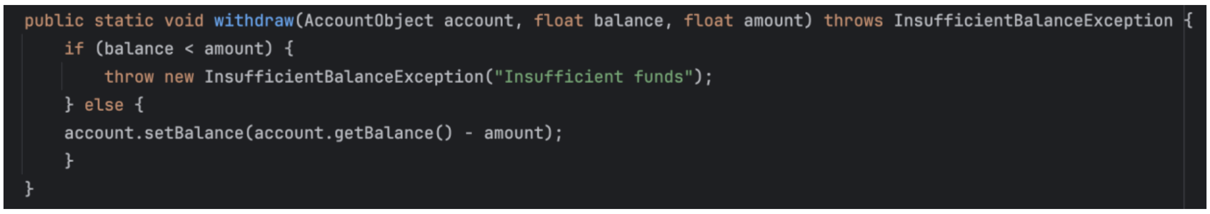
This class cleanly follows the **Single Responsibility Principle** by isolating all account-related data storage and retrieval logic in one place.



```
public class AccountRepository {  5 usages  defford *
    private static final ArrayList<AccountObject> accounts = new ArrayList<>();  4 usages

    public AccountRepository() {};  no usages  defford

    Windsurf: Refactor | Explain | Docstring | ×
    public static int nextId() { return accounts.size() + 1; }

    Windsurf: Refactor | Explain | Docstring | ×
    public static void addAccount(AccountObject account) { accounts.add(account); }

    Windsurf: Refactor | Explain | Docstring | ×
    public static AccountObject findAccountByName(String name) throws AccountNotFoundException {...}

    Windsurf: Refactor | Explain | Docstring | ×
    public static List<AccountObject> getAccounts() { return new ArrayList<>(accounts); }
}
```

Example #2: Exception Handling

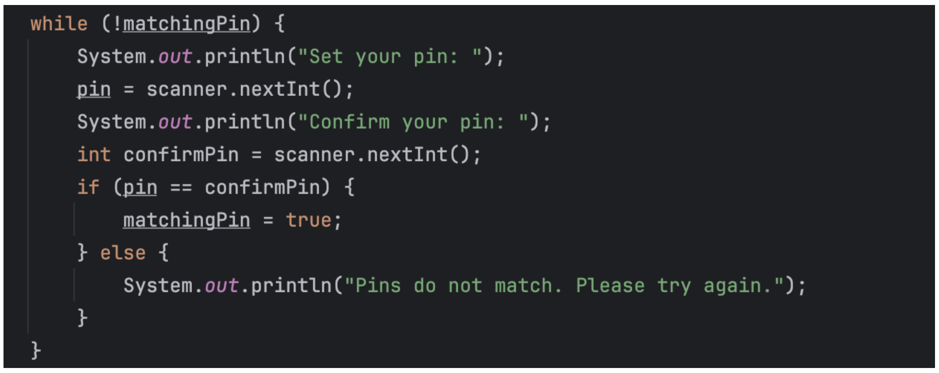
This method uses a predefined custom exception to handle insufficient balance scenarios, which enforces **fail-fast behaviour** and makes the business rule (no overdrafts) explicit.



```
public static void withdraw(AccountObject account, float balance, float amount) throws InsufficientBalanceException {
    if (balance < amount) {
        throw new InsufficientBalanceException("Insufficient funds");
    } else {
    account.setBalance(account.getBalance() - amount);
    }
}
```

Example #3: User Validation Loop

This input loop demonstrates **defensive programming** by requiring the user to confirm their PIN and validating it before proceeding.



```
while (!matchingPin) {
    System.out.println("Set your pin: ");
    pin = scanner.nextInt();
    System.out.println("Confirm your pin: ");
    int confirmPin = scanner.nextInt();
    if (pin == confirmPin) {
        matchingPin = true;
    } else {
        System.out.println("Pins do not match. Please try again.");
    }
}
```