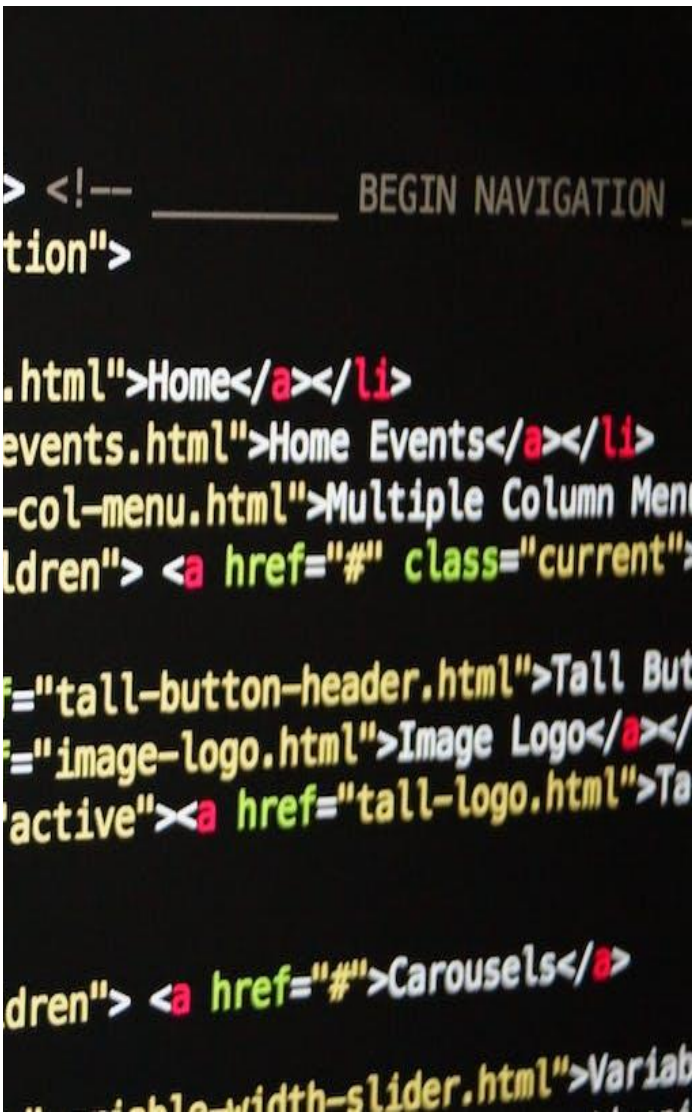
Veerain’s ATM report 2023

CS22B049



6TH OCTOBER

Review of My Project…

Automated Teller Machine (ATM) System

Overview

This ATM system is designed to provide a userfriendly interface for account holders to access their accounts, perform transactions, and for administrators to manage accounts and ATM functionalities. It adheres to key objectoriented programming (OOP) principles, including Abstraction, Encapsulation, Polymorphism, and Interfaces.

ObjectOriented Programming Principles Implementation

Abstraction

In the ATM system, abstraction is achieved through the use of classes and interfaces to abstract the various components and operations of the ATM system:

Classes: The code utilizes classes to encapsulate related data and behavior. For example, classes like **BankAtm**, **BankDataBase**, **UserInfo**, and **BankAtm** represent different aspects of the ATM system, providing a structured way to organize and manage data and operations.

Interfaces: The atmMachine interface defines a contract that ATM classes must adhere to, abstracting the essential functions that any ATM should support, such as screen interaction, keypad input, cash dispensing, and deposit handling.

Encapsulation:

Encapsulation is a fundamental principle in OOP, and it's implemented in my code to hide the internal details and protect data from unauthorized access:

Access Modifiers: Access modifiers like private and protected are used to control the visibility of class members. For instance, the UserInfo class encapsulates user data, ensuring that sensitive information like **PINs** and ***balances*** are not directly accessible.

Getters and Setters: Getter and setter methods, such as getPin() and setPin(), are used to encapsulate access to private member variables, maintaining control over how data is accessed and modified.

Polymorphism

Polymorphism allows classes to implement different code for the same method for their convenience.
My code showcases polymorphism through method overriding and interface implementation:

Method Overriding: The BankAtm class, which represents an ATM, extends the UserOperations class and overrides methods like **SetCapacity();**

Interface Implementation: The atmMachine interface defines a set of methods that all ATM classes must implement. **This promotes polymorphism, as different ATM classes can provide their unique implementations for these methods while adhering to the same interface.(this may not be that important but just to complete…)**

Interfaces

Interfaces define a contract that classes must adhere to, ensuring consistency in behavior. In my code, the atmMachine interface defines the core methods and operations that any ATM must support:

screen(BankDataBase obj): Displays messages to the user.
keyPad(int choose, UserInfo user): Receives numeric input from the user.
cashDispenser(int twoth, int fivehun, int twohun, int hun, int fif, int twen, int ten): Dispenses cash to the user.
depositSlot(UserInfo user): Handles deposit operations.

Constraints

My ATM system also adheres to specific constraints, including:

1. ATM PIN (Variable Length): The system supports ATM PINs with variable lengths, ensuring flexibility in user authentication.

2. CARDLESS ATM: While not explicitly mentioned in the code, the architecture allows for cardless ATM transactions by using account numbers and PINs for authentication.

3. DEPOSIT of Coins: The system is designed to accept coins of all denominations during deposits, enhancing its versatility.

4. Profile Update: Users can update their basic profile information at the ATM, promoting selfservice and convenience, while restricting changes to crucial account details.

Conclusion

This ATM system exemplifies the implementation of core OOP principles, making it extensible, maintainable, and robust. The use of classes, interfaces, encapsulation, method overriding, and adherence to constraints collectively create a wellstructured and functional ATM system.