COURSE: (CL-1004) OBJECT ORIENTED PROGRAMMING LAB
LAB TASK # 9

NOTE:
Only submit .cpp file of each question in a folder. Anyone who submits any other format file will get straight **ZERO.** Each question should have a separate .cpp file. Copy Paste or other UFM will also get **ZERO**. Use the following format for naming the folder Roll#_Name (P18-1234_NAME).

Q No.1: . You are tasked with developing a smart home automation system that can control various types of devices. Each device has its unique functionalities but shares some common actions such as turning on, turning off, and checking status. Your goal is to implement this system using object-oriented programming principles, particularly focusing on polymorphism.

Instructions:

Define the Base Class:

Create an abstract base class called SmartDevice with three abstract methods:

turn_on()

turn_off()

status()

Create Derived Classes:

Implement the following derived classes, each inheriting from SmartDevice and providing unique implementations for the abstract methods:

SmartLight

SmartThermostat

SmartSpeaker

Each class should have the following method implementations:

turn_on(): Print a message indicating the device is now on with a specific state.

turn_off(): Print a message indicating the device is now off.

status(): Print a message indicating the current status of the device.

Add Additional Devices:

Add two more derived classes of your choice, such as SmartDoorLock and SmartCamera. Implement their turn_on(), turn_off(), and status() methods with unique messages.

Implement the Home Automation Controller:

Create a function control_devices(devices: List[SmartDevice]) that accepts a list of SmartDevice objects. This function should:

Iterate through each device.

Call the turn_on() method.

Call the status() method.

Call the turn_off() method.

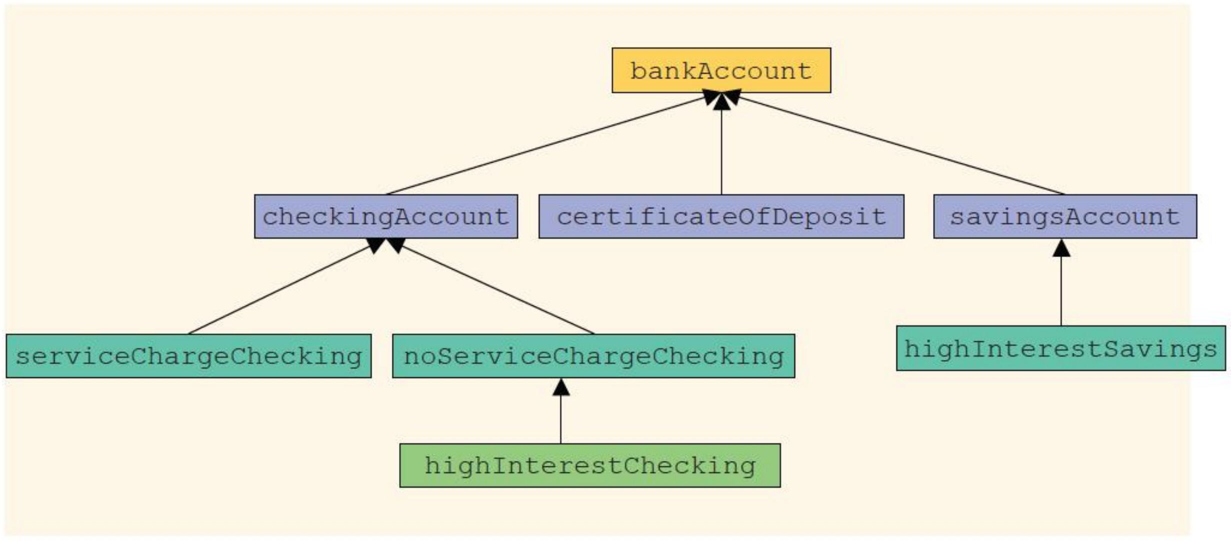
Also test your code in main().

Q No.2: Use abstract classes and pure virtual functions to design classes to manipulate various types of accounts. For simplicity, assume that the bank offers three types of accounts: savings, checking, and certificate of deposit, as described next.

Savings accounts: Suppose that the bank offers two types of savings accounts: one that has no minimum balance and a lower interest rate and another that requires a minimum balance and has a higher interest rate.

Checking accounts: Suppose that the bank offers **three types of checking accounts:** one with a monthly service charge, limited check writing, no minimum balance, and no interest; another with no monthly service charge, a minimum balance requirement, unlimited check writing and lower interest; and a third with no monthly service charge, a higher minimum requirement, a higher interest rate, and unlimited check writing.

Certificate of deposit (CD): In an account of this type, money is left for some time, and these accounts draw higher interest rates than savings or checking accounts. Suppose that you purchase a CD for six months. Then we say that the CD will mature in six months. The penalty for early withdrawal is stiff.



Note that the classes bankAccount and checkingAccount are abstract. That is, we cannot instantiate objects of these classes. The other classes in Figure are not abstract.

bankAccount: Every bank account has an account number, the name of the owner, and a balance. Therefore, instance variables such as name, accountNumber, and balance should be declared in the abstract class bankAccount. Some operations common to all types of accounts are retrieve account owner's name, account number, and account balance; make deposits; withdraw money; and create monthly statement. So include functions to implement these operations. Some of these functions will be pure virtual.

checkingAccount: A checking account is a bank account. Therefore, it inherits all the properties of a bank account. Because one of the objectives of a checking account is to be able to write checks, include the pure virtual function writeCheck to write a check.

serviceChargeChecking: A service charge checking account is a checking account. Therefore, it inherits all the properties of a checking account. For simplicity, assume that this type of account does not pay any interest, allows the account holder to write a limited number of checks each month, and does not require any minimum balance. Include appropriate named constants, instance variables, and functions in this class.

noServiceChargeChecking: A checking account with no monthly service charge is a checking account. Therefore, it inherits all the properties of a checking account. Furthermore, this type of account pays interest, allows the account holder to write checks, and requires a minimum balance.

highInterestChecking: A checking account with high interest is a checking account with nomonthly service charge. Therefore, it inherits all the properties of a no service charge checking account. Furthermore, this type of account pays higher interest and requires a higher minimum balance than the no service charge checking account.

savingsAccount: A savings account is a bank account. Therefore, it inherits all the properties of a bank account. Furthermore, a savings account also pays interest.

highInterestSavings: A high-interest savings account is a savings account.Therefore, it inherits all the properties of a savings account. It also requires a minimum balance.

certificateOfDeposit: A certificate of deposit account is a bank account. Therefore, it inherits all the properties of a bank account. In addition, it has instance variables to store the number of CD maturity months, interest rate, and the current CD month.

Write the definitions of the classes described in this programming exercise and a program to test your classes.

Note: For user understanding purposes you should write comment with each line of code.