

Write a Java Program which does the following:

Problem Description:

(Account Inheritance Hierarchy) Create an inheritance hierarchy that a bank might use to represent customers' bank accounts.

- All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts.
 - More specific types of accounts also exist.
 - Savings accounts, for instance, earn interest on the money they hold. Checking accounts, for instance, charge a fee per transaction (i.e., credit or debit).
 - Current accounts, on other hand, has overdraft(Negative balance) facility to withdraw the money.
 - Create an inheritance hierarchy containing base class Account and derived classes **SavingsAccount**, **CheckingAccount** and **CurrentAccount** that inherit from class Account.
 - Base class Account should include one data member of type double to represent the account balance.
 - The class should provide a constructor that receives an initial balance and uses it to initialize the data member.
 - The constructor should validate the initial balance to ensure that it is greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid.
 - The class should provide three member functions. Member function credit() should add an amount to the current balance.
 - Member function debit() should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance."
 - Member function getBalance() should return the current balance.
1. Derived class **SavingsAccount** should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account.
 - **SavingsAccount**'s constructor should receive the initial balance, as well as an initial value for the **SavingsAccount**'s interest rate.

- **SavingsAccount** should provide a public member function `calculateInterest` that returns a double indicating the amount of interest earned by an account.
 - Member function `calculateInterest` should determine this amount by multiplying the interest rate by the account balance.
 - [Note: **SavingsAccount** should inherit member functions `credit` and `debit` as is without redefining them.]
2. Derived class **CheckingAccount** should inherit from base class **Account** and include an additional data member of type double that represents the fee charged per transaction.
- **CheckingAccount**'s constructor should receive the initial balance, as well as a parameter indicating a fee amount.
 - Class **CheckingAccount** should redefine member functions `credit` and `debit` so that they subtract the fee from the account balance whenever either transaction is performed successfully.
 - **CheckingAccount**'s versions of these functions should invoke the base-class **Account** version to perform the updates to an account balance.
 - **CheckingAccount**'s `debit` function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance).
 - [Hint: Define **Account**'s `debit` function so that it returns a boolean indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]
3. Derived class **CurrentAccount** should inherit the functionality of an **Account**, but also include a data member of type double indicating the maximum overdraft limit (percentage) assigned to the **Account**.
- **CurrentAccount**'s constructor should receive the initial balance, as well as an initial value for the **CurrentAccount**'s maximum overdraft limit.
 - **CurrentAccount** should redefine member function `debit`, so that the account holder can withdraw money upto a maximum of overdraft limit if the balance is below 0.
 - [Note: An overdraft occurs when withdrawals from a bank account exceed the available balance which gives the account a negative balance]
4. After defining the classes in this hierarchy, write a Test program that creates objects of each class and tests their member functions.
- For Example:
- Add interest to the **SavingsAccount** object by first invoking its `calculateInterest` function, then passing the returned interest amount to the object's `credit` function.