

Challenge 3: Implement a Banking Application Using Polymorphism

In this challenge, you have to implement a basic banking application by implementing the Account class along with two derived classes, SavingsAccount and CheckingAccount.

WE'LL COVER THE FOLLOWING ^

- Problem Statement
 - Input
 - Output
 - Sample Input
 - Sample Output
- Coding Exercise

Problem Statement

Write a code that has:

- A **base class** named `Account`.
 - Inside it *define*:
 - A field, `private double _balance`
 - A `protected` property, `Balance`, to access the balance
 - `public virtual bool Withdraw(double amount)`
 - `public virtual bool Deposit(double amount)`
 - `public virtual void PrintBalance()`
- Then, there are **two derived classes**
 - `SavingsAccount` class has:
 - A `private` field `_interestRate` set to **0.8**

- An overridden `Withdraw()` method that deducts *amount* from *balance* with *interestRate* only if enough balance is available. The withdrawal amount should be greater than zero.
 - Returns `true` if the transaction was successful and `false` otherwise.
- An overridden `Deposit()` method that adds *amount* to *balance* with *interestRate* and also checks if the amount to be deposited is greater than zero
 - Returns `true` if the transaction was successful and `false` otherwise.
- `PrintBalance()` displays the balance in the *account*
- `CheckingAccount` class has:
 - An overridden `Withdraw()` method that deducts *amount* from *balance* only if enough balance is available and the withdrawal amount should be greater than zero
 - Returns `true` if the transaction was successful and `false` otherwise.
 - An overridden `Deposit()` method that adds *amount* in *balance* and also checks if the amount to be deposited is greater than zero
 - Returns `true` if the transaction was successful and `false` otherwise.
 - `PrintBalance()` displays the balance in the *account*

For SavingsAccount:

*WithdrawalFormula : Balance = Balance - (amount + (amount * interestRate))*

*DepositFormula : Balance = Balance + (amount + (amount * interestRate))*

Input #

- In the `Savings` class, `Balance` is set to `5000` using the `base()` parametrized constructor
- In the `Current` class, `Balance` is set to `5000` using the `base()` parametrized constructor

Output

- Balance before withdrawal from the savings account
- Balance after withdrawal from the savings account
- Balance before withdrawal from the current account
- Balance after withdrawal from the current account

Sample Input

```
Account SAccount = new SavingsAccount(5000);

// creating saving account object
SAccount.Deposit(1000);
SAccount.PrintBalance();

SAccount.Withdraw(3000);
SAccount.PrintBalance();

// creating checking account object
Account CAccount = new CheckingAccount(5000);
CAccount.Deposit(1000);
CAccount.PrintBalance();

CAccount.Withdraw(3000);
CAccount.PrintBalance();
```

Sample Output

```
The saving account balance is: 6800
The saving account balance is: 1400
```

The checking account balance is: 6000


The checking account balance is: 3000

Coding Exercise

First, take a close look and design a step-by-step algorithm before jumping to the implementation. This problem is designed for practice, so try to solve it on your own. If you get stuck, you can always refer to the solution provided in the solution review.

Good luck!

Note: Uncomment the block written in the `Main()` method to test your code.

 Exercise

 Solution

```
// Implement classes here

class Demo {

    public static void Main(string[] args) {
        //Uncomment the below block to test your code
        /*
        Account SAccount = new SavingsAccount(5000);

        // Creating saving account object
        SAccount.Deposit(1000);
        SAccount.PrintBalance();

        SAccount.Withdraw(3000);
        SAccount.PrintBalance();

        Console.WriteLine();

        // Creating checking account object
        Account CAccount = new CheckingAccount(5000);
        CAccount.Deposit(1000);
        CAccount.PrintBalance();

        CAccount.Withdraw(3000);
        CAccount.PrintBalance();
        */
    }
}
```



The solution will be explained in the next lesson.