Solution Review: Implement Banking Application Using Polymorphism

This review provides a detailed analysis on how to solve the 'Implement a Banking Application Using Polymorphism' challenge.

WE'LL COVER THE FOLLOWING ^

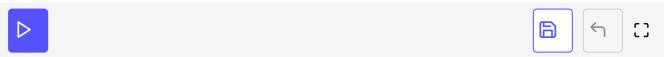
- Solution
 - Explanation

Solution

```
class Account {
   private double _balance;
   protected double Balance {
       get { return this._balance; }
       // Check before setting the balance
       set { if(value>=0)
              this._balance = value;
   // Constructor
   public Account(double balance) {
       this.Balance = balance;
   //Virtual Methods
   public virtual bool Deposit(double amount) {
        return false;
   }
   public virtual bool Withdraw(double amount) {
       return false;
   }
   public virtual void PrintBalance() {
       Console.WriteLine("The balance is: " + Balance);
```

```
}
class SavingsAccount : Account {
    private double _interestRate;
    // Constructor
    public SavingsAccount(double balance)
        : base(balance)
      // It's always preferable to initialize fields inside a constructor
        this._interestRate = 0.8;
    // Overridden Methods
    public override bool Deposit(double amount) {
        if(amount > 0) { // Check if amount is non-zero and non-negative
            // Adding to balance with interest rate
            Balance += amount + (amount * this._interestRate);
            return true;
        return false;
    }
    public override bool Withdraw(double amount) {
        if(amount > 0 && amount <= Balance) { // Check if amount is non-zero and less than eq
            // Deducting from balance with interest rate
            Balance -= amount + (amount * this._interestRate);
            return true;
        return false;
    }
    public override void PrintBalance() {
        Console.WriteLine("The saving account balance is: " + base.Balance);
    }
}
class CheckingAccount : Account
    // Constructor
    public CheckingAccount(double balance)
        : base(balance) { }
    // Overridden Methods
    public override bool Deposit(double amount) {
        if (amount > 0)
        {
            Balance += amount;
            return true;
        return false;
    }
    public override bool Withdraw(double amount) {
        if (amount > 0 && amount <= Balance)</pre>
        {
            Balance -= amount;
            return true:
```

```
}
        return false;
    }
    public override void PrintBalance() {
        Console.WriteLine("The checking account balance is: " + base.Balance);
    }
}
class Demo {
    public static void Main(string[] args) {
        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();
    }
}
```



Explanation

- We have implemented the Account class which has the _balance double field, and three public virtual methods Deposit(double amount),
 Withdraw(double amount) and PrintBalance().
- Implemented SavingsAccount and CheckingAccount classes derived from the Account class.
- SavingsAccount class has the private double _interestRate field and following methods:

Overwiding with the (to bloom of the deducte ground

- from the *balance* with *interestRate* after checking the validity of a transaction.
- Overriding Deposit(double amount) method that adds *amount* in the *balance* with *interestRate* after checking the validity of a transaction.
- PrintBalance() displays the balance in the *account*.
- CheckingAccount class has the following methods:
 - Overriding Withdraw(double amount) that deducts *amount* from *balance* after checking the validity of a transaction.
 - Overriding Deposit(double amount) that adds *amount* in *balance* after checking the validity of a transaction.
 - PrintBalance() displays the balance in the *account*.
- Created SavingsAccount and CheckingAccount objects by calling the parameterized constructors of the classes and printed their balance by calling their respective PrintBalance() methods