

po-2021-22L-wykł-kmolenda

Uczestnicy

Kompetencje

Oceny

Zaliczenie przedmiotu

Podręczniki, kursy, materiały

Repetitorium

Konwencje kodowania w C# i dokumentowanie kodu

System typów języka C# (i platformy .Net)

Modelowanie obiektowe w C#

Well formed types

Kokpit

Strona główna

Kalendarz

Prywatne pliki

Diagram UML przedstawiający hierarchię klas:

```
graph TD; IAccount[IAccount Interface] --> Account[Account Class]; IAccount --> IAccountWithLimit[IAccountWithLimit Interface]; Account --> AccountPlus[AccountPlus Class]; Account --> Account[Account Class];
```

Opis diagramu: Klasa `IAccount` ma dwie implementacje: `Account` i `IAccountWithLimit`. Klasa `Account` ma dziesięć metod: `Balance`, `Blocked`, `Name`, `Methods`, `Account`, `Block`, `Deposit`, `IsBlocked`, `Withdraw`, `Unblock` i `Withdrawal`. Klasa `IAccountWithLimit` ma dwa dodatkowe właściwości: `AvailableFounds` i `OneTimeDebetLimit`.

Przykładowy kod źródłowy (już osadzone w zadaniu):

```
// file: IAccount.cs
namespace Bank
{
    public interface IAccount
    {
        // nasza klientka, bez spacji przed i po
        // readonly - modyfikowalna wyłącznie w konstruktorze
        string Name {get; }

        // bilans, aktualny stan środków, podawany w zaokrągleniu do 2 miejsc po przecinku
        decimal Balance {get; }

        // czy konto jest zablokowane
        bool IsBlocked {get; }
        void Block(); // zablokowanie konta
        void Unblock(); // odzablokowanie konta

        // wpłata, dla kwoty ujemnej - ignorowana (false)
        // jeśli konto zablokowane - ignorowana (false)
        // true jeśli wpłacono i nastąpiła zmiana salda
        bool Deposit(decimal amount);

        // wpłata, dla kwoty ujemnej - ignorowana (false)
        // jeśli konto zablokowane - ignorowana (false)
        // jeśli wpłata przekracza ilość środków - ignorowana (false)
        // true jeśli wpłata i nastąpiła zmiana salda
        bool Withdraw(decimal amount);
    }
}

// file: IAccountWithLimit.cs
namespace Bank
{
    public interface IAccountWithLimit : IAccount
    {
        // przyznany limit debetowy
        // możliwość zmiany, jeśli konto nie jest zablokowane
        decimal OneTimeDebetLimit {get; set; }

        // dostępne środki, z uwzględnieniem limitu
        decimal AvailableFounds {get; }
    }
}

// file: Account.cs
namespace Bank
{
    public class Account : IAccount
    {
        protected const int PRECISION = 4;

        public string Name {get; }
        public decimal Balance {get; private set; }

        public bool IsBlocked {get; private set;} = false;
        public void Block() => IsBlocked = true;
        public void Unblock() => IsBlocked = false;

        public Account(string name, decimal initialBalance = 0)
        {
            if (name == null || initialBalance < 0)
                throw new ArgumentException();
            Name = name.Trim();
            if (Name.Length < 3)
                throw new ArgumentException();
            Balance = Math.Round(initialBalance, PRECISION);
        }

        public bool Deposit(decimal amount)
        {
            if (amount <= 0 || IsBlocked) return false;
            Balance = Math.Round(Balance + amount, PRECISION);
            return true;
        }

        public bool Withdraw(decimal amount)
        {
            if (amount <= 0 || IsBlocked || amount > Balance) return false;
            Balance = Math.Round(Balance - amount, PRECISION);
            return true;
        }

        public override string ToString()
        {
            IsBlocked ? $"Account name: {Name}, balance: {Balance:F2}, blocked={true}" :
            $"Account name: {Name}, balance: {Balance:F2}";
        }
    }
}
```

UWAGA: Piszesz tylko kod klasy `AccountPlus`.

Na przykład:

Test	Wynik
1) Tworzenie konta, zadanie limatu // utworzenie konta John z domylnym limitem 100 var John = new AccountPlus("John");	Account name: John, balance: 0.00, available: 100.00 Account name: John, balance: 200.00, available: 200.00
var Eve = new AccountPlus("Eve");	Account name: Eve, balance: 99.00, available: 210.00 Account name: Eve, balance: 99.00, blocked, available: 210.00, limit: 120.00

```

// utworzenie konta plus z ustawionym limitem na 200 i bilansem początkowym 99
var eve = new AccountPlus("eve", initialLimit: 200.0m, initialBalance: 99.0m);
Console.WriteLine(eve);

// zmiana limitu, konto nie zablokowane
eve.OneTimeDebetLimit = 120.0m;
Console.WriteLine(eve);

// próba zmiany limitu, konto zablokowane
eve.Block();
eve.OneTimeDebetLimit = 500.0m;
Console.WriteLine(eve);

// próba utworzenia konta z limitem ujemnym
var stan = new AccountPlus(name: "stan", initialLimit: -200.0m);
Console.WriteLine(stan);

// scenario: wpłaty wpłaty, blokady konta
// utworzenie konta plus z domyslnym limitem 100
var john = new AccountPlus("john", initialBalance: 100.0m);
Console.WriteLine(john);

// wpłata - podanie kwoty ujemnej
{john.Withdrawal(-50.0m);}
Console.WriteLine(john);

// wpłata bez wykorzystania debetu
{john.Withdrawal(50.0m);}
Console.WriteLine(john);

// wpłata z wykorzystaniem debetu
{john.Withdrawal(100.0m);}
Console.WriteLine(john);

// konto zablokowane, wpłaty niemożliwe
{john.Withdrawal(10.0m);}
Console.WriteLine(john);

// wpłata odblokowująca konto
{john.Deposit(10.0m);}
Console.WriteLine(john);

// wpłata podanie kwoty ujemnej
{john.Deposit(-80.0m);}
Console.WriteLine(john);

```

Odpowiedź: (System kar: 0 %)

Zresetuj odpowiedź

```

21     private decimal StdLimit { get; set; }
22     public decimal OneTimeDebetLimit
23     {
24         get
25         {
26             return _limit;
27         }
28         set
29         {
30             if (value > 0 && !IsBlocked)
31             {
32                 _limit = value;
33                 StdLimit = value;
34             }
35         }
36     }
37
38     public new bool Deposit(decimal amount)
39     {
40         if (amount > 0)
41         {
42             if (OneTimeDebetLimit != StdLimit)
43             {
44                 if (amount + OneTimeDebetLimit >= StdLimit)
45                 {
46                     base.Unblock();
47                     decimal brakujace = StdLimit - OneTimeDebetLimit;
48                     amount -= Math.Round(brakujace, PRECISION);
49                     _limit = StdLimit;
50                     base.Deposit(Math.Round(amount, PRECISION));
51                 }
52             }
53         }
54         else
55         {
56             base.Deposit(Math.Round(amount, PRECISION));
57         }
58
59         if (OneTimeDebetLimit >= StdLimit)
60             base.Unblock();
61
62         return true;
63     }
64     return false;
65 }
66 public new bool Withdrawal(decimal amount)
67 {
68     if (amount > 0 && IsBlocked == false)
69     {
70         if (AvailableFounds > Math.Round(amount, PRECISION))
71         {
72             if (Balance < Math.Round(amount, PRECISION))
73             {
74                 decimal wartoscWyplaty = Balance;
75                 base.Withdrawal(Math.Round(Balance, PRECISION));
76                 amount = Math.Round(wartoscWyplaty, PRECISION);
77                 _limit -= Math.Round(amount, PRECISION);
78                 base.Block();
79             }
80         }
81         else
82         {
83             base.Withdrawal(Math.Round(amount, PRECISION));
84             return true;
85         }
86     }
87     return false;
88 }
89
90     public decimal AvailableFounds => OneTimeDebetLimit + Balance;
91     public new void Unblock()
92     {
93         if (OneTimeDebetLimit >= StdLimit)
94         {
95             base.Unblock();
96         }
97     }
98 }
99
100    public override string ToString()
101    {
102        if (IsBlocked)
103        {
104            return $"Account name: {Name}, balance: {Balance:F2}, blocked, available founds: {AvailableFounds:F2}, limit: {StdLimit:F2}";
105        }
106        else
107    }

```

Sprawdź

Test	Oczekiwane	Otrzymane
<p>✓ // weryfikacja, czy implementowany jest interfejs IAccountWithLimit</p> <pre> /* var account = new AccountPlus("AAA", 100.00m); if (account is Bank.IAccountWithLimit) Console.WriteLine("AccountWithLimit implemented"); else Console.WriteLine("AccountWithLimit not implemented"); // weryfikacja, czy dziedziczeno jest klasę Account */ if (account is Bank.Account) Console.WriteLine("Account inherited"); else Console.WriteLine("Account not inherited"); </pre>	IAccountWithLimit implemented Account inherited	IAccountWithLimit implemented Account Inherited
<p>✓ // tworzenie konta, zmiana limtu</p> <pre> // utworzenie konta plus z domyslnym limitem 100 var john = new AccountPlus("John"); Console.WriteLine(john); // utworzenie konta plus z ustawionym limitem na 200 i bilansem początkowym 99 var eve = new AccountPlus("eve", initialLimit: 200.0m, initialBalance: 99.0m); Console.WriteLine(eve); // zmiana limitu, konto nie zablokowane eve.OneTimeDebetLimit = 120.0m; Console.WriteLine(eve); // próba zmiany limitu, konto zablokowane eve.Block(); eve.OneTimeDebetLimit = 500.0m; Console.WriteLine(eve); // próba utworzenia konta z limitem ujemnym var stan = new AccountPlus(name: "Stan", initialLimit: -200.0m); Console.WriteLine(stan); </pre>	Account name: John, balance: 0.00, available founds: 100.00, limit: 100.00 Account name: Eve, balance: 99.00, available founds: 200.00, limit: 200.00 Account name: Eve, balance: 99.00, available founds: 210.00, limit: 120.00 Account name: Eve, balance: 99.00, blocked, available founds: 210.00, limit: 120.00 Account name: Stan, balance: 0.00, available founds: 0.00, limit: 100.00	Account name: John, balance: 0.00, available founds: 100.00, limit: 100.00 Account name: Eve, balance: 99.00, available founds: 200.00, limit: 200.00 Account name: Eve, balance: 99.00, available founds: 210.00, limit: 120.00 Account name: Eve, balance: 99.00, blocked, available founds: 210.00, limit: 120.00 Account name: Stan, balance: 0.00, available founds: 0.00, limit: 100.00

```

Console.WriteLine("start");

✓ // scenariusz: wpłata wpłaty, blokada konta
// utworzenie konta plus z domyslnym limitem 100
var john = new AccountPlus("John", initialBalance: 100.00);
Console.WriteLine(john);

// wpłata - podanie kwoty ujemnej
john.Withdrawal(-50.00);
Console.WriteLine(john);

// wpłata bez wykorzystaniem debetu
john.Withdrawal(20.00);
Console.WriteLine(john);

// konto zablokowane, wpłata niemożliwa
john.Withdrawal(100.00);
Console.WriteLine(john);

// wpłata odblokowująca konto
john.Deposit(80.00);
Console.WriteLine(john);

// wpłata podanie kwoty ujemnej
john.Deposit(-80.00);
Console.WriteLine(john);

✓ // sytuacje specjalne
// konto z zerową stanem
var account = new AccountPlus("John", initialBalance: 0, initialLimit: 0);
Console.WriteLine(account);
account.Withdrawal(100);
Console.WriteLine(account);

// zero saldo, limit 50
account.Deposit(50);
account.Withdrawal(50);
Console.WriteLine(account);
account.Withdrawal(0); // zero wpłata
Console.WriteLine(account);
account.Unblock(); // próba odblokowania konta
Console.WriteLine(account);
account.Deposit(100); // likwidacja debetu, zeroowy bilans
Console.WriteLine(account);

```

Przeszedły wszystkie testy! ✓

Przepisane

Punkty dla tej odpowiedzi: 5.00/5.00.

◀ Zadanie: Konto w banku (part 1)

Przejdz do...

Zadanie: implementacja uproszczonego stosu generycznego ▶

Zapis podjęcie ..

Jestes zalogowany(a) jako Piotr Krystian (Wyloguj)
po-2021-22-wyklad-kmolecka
Podsumowanie zasad ercachowania danych