

# 2020/21 - Wprowadzenie do programowania (C#) - LAB (K. Molenda)

Kokpit / Moje kursy / wdp-2020-21-lab-kmolenda / Lab. 09-10. OOP, interfejsy / Zadanie: Konto w banku (part 2)

Rozpoczęto	Sunday, 13 June 2021, 01:02 AM
Stan	Ukończone
Ukończono	Sunday, 13 June 2021, 14:18 PM
Wykorzystany czas	13 godzin 16 min.
Ocena	5,00 pkt. na 5,00 pkt. możliwych do uzyskania (100%)

Pytanie 1
Poprawne
Ocena: 5,00 z 5,00
Oglądz pytanie

Zadanie jest kontynuacją kroku pierwszego (Klasa `Account` - step 1).

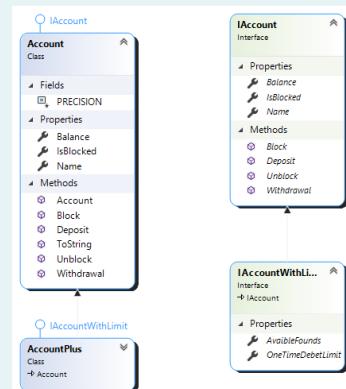
Tworzysz bibliotekę klas. Pracujesz w przestrzeni nazw `Bank`.

- Dany jest interfejs `IAccount` określający wymagania dotyczące klasy `Account`.
- Dana jest klasa `Account` implementująca interfejs `IAccount` i spełniającą wymagania opisane w kroku 1.
- Dany jest interfejs `IAccountWithLimit`, rozszerzający koncepcję konta bankowego o tzw. "jednorazowy limit debetowy" (`OneTimeDebetLimit`):
  - Ustalony jest indywidualnie podczas tworzenia konta (domyślnie 100, wartość nieujemna - w przypadku podania wartości ujemnej limit wynosi 0),
  - Limit można zmienić dla już istniejącego konta (zwiększenie lub zmniejszenie), ale tylko wtedy, gdy konto nie jest zablokowane. Przypisanie wartości ujemnej nie generuje wyjątku, operacja jest nieskuteczna, limit nie ulega zmianie.
  - Umozliwia on jednorazowe zrealizowanie wypłaty przekraczającej aktualny stan środków (tzw. debet), ale nie przekraczającą kwoty ustalonego "jednorazowego limitu debetowego".
  - Po wykonaniu takiej transakcji (przekraczającej bieżące środki), ale w ramach przyznanego limitu, konto zostaje automatycznie zablokowane.
  - Dla konta zablokowanego nie można wykonać transakcji wypłaty, można natomiast dokonać niezerowej wypłaty.
  - Konto zostanie automatycznie odzłokowane po wykonaniu wypłaty (wpłaty), które zlikwidują debet. Przywrócona zostanie wtedy również funkcjonalność "jednorazowego limitu debetowego".
  - Zarówno wypłata jak i wpłata (metody `Deposit` oraz `Withdrawal`) nie zgłasza wyjątków, podobnie jak w klasie `Account` - w sytuacji niedozwolonej operacji nie wykonują jej, raportując wykonanie lub niewykonanie zwracaną wartością `true` lub `false`.

Utwórz klasę `AccountPlus`, dziedziczącą z klasą `Account` i implementującą interfejs `IAccountWithLimit`. Przymij dodatkowe założenia:

- W każdym momencie konta można "ręcznie" zablokować (metoda `Block()`). Konto zablokowane można "ręcznie" odblokować (metoda `Unblock()`) jedynie wtedy, gdy nie ma zadłużenia w ramach "jednorazowego limitu debetowego". Konto "ręcznie" zablokowane odblokowuje się automatycznie przy operacji wypłaty powodujączej, że nie ma zadłużenia.
- Tekstowa reprezentacja konta plus jest napis o formacie:
  - jeśli konto nie jest zablokowane: `Account name: {Name}, balance: {Balance}, available funds: {AvailableFounds}, limit: {OneTimeDebetLimit}`
  - jeśli konto jest zablokowane: `Account name: {Name}, balance: {Balance}, blocked, , available funds: {AvailableFounds}, limit: {OneTimeDebetLimit}` Saldo, dostępne środki, limit podawane są w zaokrągleniu do 2. miejsca po przecinku.
- Wszystkie operacje arytmetyczne wykonywane z dokładnością do 4. cyfr po przecinku.

Przy realizacji klas kieruj się testami weryfikującymi poprawność Twojej implementacji.



## Kody źródłowe (już osadzone w zadaniu)

```

// file: IAccount.cs
namespace Bank
{
    public interface IAccount
    {
        // nazwa klienta, 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(); // odblokowanie konta

        // wypłata, dla kwoty ujemnej - zignorowana (false)
        // jeśli konto zablokowane - zignorowana (false)
        // true jeśli wykonano i nastąpiła zmiana salda
        bool Deposit(decimal amount);

        // wypłata, dla kwoty ujemnej - zignorowana (false)
        // jeśli konto zablokowane - zignorowana (false)
        // jeśli jest niedostarczająca ilość środków - zignorowana (false)
        // true jeśli wykonano i nastąpiła zmiana salda
        bool Withdrawal(decimal amount);
    }
}

// file: IAccountWithLimit.cs
namespace Bank
{
    public interface IAccountWithLimit : IAccount
    {
        // przyznaný 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
    {
    }
}
  
```

Nawigacja w teście



Zakończ przegląd

```

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 ArgumentOutOfRangeException();
    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 Withdrawal(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"
              : $"Account name: {Name}, balance: {Balance:F2}";
}

```

**UWAGA:** Piszesz tylko kod klasy **AccountPlus**.

Na przykład:

Test	Wynik
// tworzenie konta, zmiana limitu // 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);  // scenariusz: wplaty, blokada konta // utworzenie konta plus z domyslnym limitem 100 var john = new AccountPlus("John", initialBalance: 100.0m); Console.WriteLine(john);  // wypłata - podanie kwoty ujemnej john.Withdrawal(-50.0m); Console.WriteLine(john);  // wypłata bez wykorzystania debetu john.Withdrawal(50.0m); Console.WriteLine(john);  // wypłata z wykorzystaniem debetu john.Withdrawal(100.0m); Console.WriteLine(john);  // konto zablokowane, wypłata niemożliwa john.Withdrawal(10.0m); Console.WriteLine(john);  // wypłata odblokowujaca konto john.Deposit(80.0m); Console.WriteLine(john);  // wpłata podanie kwoty ujemnej john.Deposit(-80.0m); Console.WriteLine(john);	Account name: John, balance: 0.00, available finds: 100.00, limit: 100.00 Account name: Eve, balance: 99.00, available finds: 299.00, limit: 200.00 Account name: Eve, balance: 99.00, available finds: 219.00, limit: 120.00 Account name: Eve, balance: 99.00, blocked, available finds: 219.00, limit: 120.00 Account name: Stan, balance: 0.00, available finds: 0.00, limit: 0.00  Account name: John, balance: 100.00, available finds: 200.00, limit: 100.00 Account name: John, balance: 100.00, available finds: 200.00, limit: 100.00 Account name: John, balance: 50.00, available finds: 150.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available finds: 50.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available finds: 50.00, limit: 100.00 Account name: John, balance: 30.00, available finds: 130.00, limit: 100.00 Account name: John, balance: 30.00, available finds: 130.00, limit: 100.00

Odpowiedź: (system kar: 0 %)

Zresetuj odpowiedź

```

1+ using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Bank
8+
9
10 class AccountPlus : Account, IAccountWithLimit
11 {
12     private decimal Limit;
13     private decimal Xlimit;
14
15     public new decimal Balance { get; private set; }
16     public new bool IsBlocked { get; private set; } = false;
17     public new void Unblock() { if (Xlimit <= Limit) { IsBlocked = false; } }
18     public new void Block() { IsBlocked = true; }
19
20     public AccountPlus(string name, decimal initialBalance = 0, decimal initialLimit = 100) : base(name, initialBalance)
21     {
22         Balance = initialBalance;
23         if (initialLimit < 0)
24         {
25             Xlimit = 0;
26             this.OneTimeDebetLimit = 0;
27         }
28         else
29         {
30             Xlimit = initialLimit;
31             this.OneTimeDebetLimit = initialLimit;
32         }
33     }
34
35     public void Deposit(decimal amount)
36     {
37         if (amount <= 0 || IsBlocked) return;
38
39         Balance = Math.Round(Balance += amount, PRECISION);
40
41         if (Balance > Limit)
42         {
43             IsBlocked = true;
44             Xlimit = Balance;
45         }
46     }
47
48     public void Withdrawal(decimal amount)
49     {
50         if (amount <= 0 || IsBlocked || amount > Balance) return;
51
52         Balance = Math.Round(Balance -= amount, PRECISION);
53
54         if (Balance < Xlimit)
55         {
56             IsBlocked = true;
57             Xlimit = Balance;
58         }
59     }
60
61     public void Block()
62     {
63         IsBlocked = true;
64         Xlimit = 0;
65     }
66
67     public void Unblock()
68     {
69         IsBlocked = false;
70         Xlimit = 0;
71     }
72
73     public override string ToString() =>
74         IsBlocked ? $"Account name: {Name}, balance: {Balance:F2}, blocked"
75                   : $"Account name: {Name}, balance: {Balance:F2}";
76 }

```

```

33     }
34
35     public decimal OneTimeDebetLimit
36     {
37         get
38         {
39             return Limit;
40         }
41         set
42         {
43             if (!IsBlocked)
44             {
45                 if (Limit != Xlimit)
46                 {
47                     var Z = value - Xlimit;
48                     Limit += Z;
49                 }
50
51                 Limit = value;
52                 Xlimit = value;
53             }
54         }
55     }
56
57     public void SprawdzLimit()
58     {
59
60         if (Xlimit > Limit)
61             Block();
62         else
63             Unblock();
64     }
65
66     public new bool Withdrawal(decimal amount)
67     {
68         if (IsBlocked == true)
69             return false;
70         else if (amount <= 0)
71             return false;
72         else if (AvailableFounds > amount)
73         {
74             if (decimal.Round(amount, PRECISION) > decimal.Round(Balance, PRECISION))
75             {
76                 amount -= decimal.Round(Balance, PRECISION);
77                 Balance -= decimal.Round(Balance, PRECISION);
78
79                 Limit -= decimal.Round(amount, PRECISION);
80                 amount -= decimal.Round(amount, PRECISION);
81                 SprawdzLimit();
82             }
83             else if (decimal.Round(amount, PRECISION) == decimal.Round(Balance, PRECISION))
84             {
85

```

Test	Oczekiwane	Wykonane
✓ /* weryfikacja, czy implementowany jest interfejs IAccountWithLimit */ */ var account = new AccountPlus("AAA", 100.00m); if (account is Bank.IAccountWithLimit)     Console.WriteLine("IAccountWithLimit implemented"); else     Console.WriteLine("IAccountWithLimit not implemented"); /* weryfikacja, czy dziedziczona jest klasa Account */ if (account is Bank.Account)     Console.WriteLine("Account inherited"); else     Console.WriteLine("Account not inherited");	IAccountWithLimit implemented Account inherited	IAccountWithLimit implemented Account inherited
✓ // tworzenie konta, zmiana limitu // 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 poczatkowym 99 var eve = new AccountPlus("Eve", initialLimit: 200.0m, initialBalance: 99.0m); Console.WriteLine(eve);  // zmiana limitu, konto nie заблоковане eve.OneTimeDebetLimit = 120.0m; Console.WriteLine(eve);  // próba zmiany limitu, konto заблоковане 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);	Account name: John, balance: 0.00, available founds: 100.00, limit: 100.00 Account name: Eve, balance: 99.00, available founds: 299.00, limit: 200.00 Account name: Eve, balance: 99.00, available founds: 219.00, limit: 120.00 Account name: Eve, balance: 99.00, blocked, available founds: 219.00, limit: 120.00 Account name: Stan, balance: 0.00, available founds: 0.00, limit: 0.00	Account name: John, balance: 0.00, available founds: 100.00, limit: 100.00 Account name: Eve, balance: 99.00, available founds: 299.00, limit: 200.00 Account name: Eve, balance: 99.00, available founds: 219.00, limit: 120.00 Account name: Eve, balance: 99.00, blocked, available founds: 219.00, limit: 120.00 Account name: Stan, balance: 0.00, available founds: 0.00, limit: 0.00
✓ // scenariusz: wpłata wypłaty, blokada 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 заблоковане, wpłata niemożliwa john.Withdrawal(10.0m); Console.WriteLine(john);  // wpłata odblokowująca konto john.Deposit(80.0m); Console.WriteLine(john);  // wpłata podanie kwoty ujemnej john.Deposit(-80.0m); Console.WriteLine(john);	Account name: John, balance: 100.00, available founds: 200.00, limit: 100.00 Account name: John, balance: 100.00, available founds: 200.00, limit: 100.00 Account name: John, balance: 50.00, available founds: 150.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available founds: 50.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available founds: 50.00, limit: 100.00 Account name: John, balance: 30.00, available founds: 130.00, limit: 100.00 Account name: John, balance: 30.00, available founds: 130.00, limit: 100.00	Account name: John, balance: 100.00, available founds: 200.00, limit: 100.00 Account name: John, balance: 100.00, available founds: 200.00, limit: 100.00 Account name: John, balance: 50.00, available founds: 150.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available founds: 50.00, limit: 100.00 Account name: John, balance: 0.00, blocked, available founds: 50.00, limit: 100.00 Account name: John, balance: 30.00, available founds: 130.00, limit: 100.00 Account name: John, balance: 30.00, available founds: 130.00, limit: 100.00
✓ // sytuacje specjalne // konto z zerowym stanem var account = new AccountPlus("John", initialBalance: 0, initialLimit: 0); Console.WriteLine(account); account.Withdrawal(10); Console.WriteLine(account);  // zerowe saldo, limit 50 account.OneTimeDebetLimit = 50; Console.WriteLine(account); account.Withdrawal(0); // zerowa wpłata Console.WriteLine(account); account.Withdrawal(10); // wpłata w ramach debetu	Account name: John, balance: 0.00, available founds: 0.00, limit: 0.00 Account name: John, balance: 0.00, available founds: 0.00, limit: 0.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00 Account name: John, balance: 0.00, blocked, available founds: 40.00, limit: 50.00 Account name: John, balance: 0.00, blocked, available founds: 40.00, limit: 50.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00	Account name: John, balance: 0.00, available founds: 0.00, limit: 0.00 Account name: John, balance: 0.00, available founds: 0.00, limit: 0.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00 Account name: John, balance: 0.00, blocked, available founds: 40.00, limit: 50.00 Account name: John, balance: 0.00, blocked, available founds: 40.00, limit: 50.00 Account name: John, balance: 0.00, available founds: 50.00, limit: 50.00

```
Console.WriteLine(account);
account.Unblock(); // próba odblokowania konta
Console.WriteLine(account);
account.Deposit(10); // likwidacja debetu, zerowy bilans
Console.WriteLine(account);
```

Przeszedł wszystkie testy! ✓

Poprawne

Punkty dla tej odpowiedzi: 5,00/5,00.

Zakończ przegląd

→ Zadanie: Konto w banku (part 1)

Przejdz do...

Zadanie: Implementacja uproszczonego stosu generycznego ▶

Jesteś zalogowany(a) jako Petek Krystian (Wyloguj)  
wdp-2020-21-lab-kmolenda  
Podsumowanie zasad przechowywania danych

