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

Rozpoczęto	Thursday, 3 June 2021, 13:02 PM
Stan	Ukończone
Ukończono	Sunday, 13 June 2021, 01:00 AM
Wykorzystany czas	9 dni 11 godzin
Ocena	**5,00** pkt. na 5,00 pkt. możliwych do uzyskania (**100**%)

Nawigacja w teście

1

Zakończ przegląd

Pytanie **1**

Poprawnie

Ocena: 5,00 z 5,00

Oflaguj pytanie

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

Dany jest interfejs `IAccount` określający wymagania dotyczące projektowanej przez Ciebie klasy `Account`.

```
using System;

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

        // wpłata, 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 niewystarczająca ilość środków - zignorowana (false)
        // true jeśli wykonano i nastąpiła zmiana salda
        bool Withdrawal(decimal amount);
    }
}
```

Nie kopiuj interfejsu, jest on dołączany podczas kompilacji!

Utwórz klasę `Account` implementującą interfejs `IAccount` i spełniającą następujące wymagania:

Dane:

- `Name` - `string`, nazwa klienta, bez spacji przed i po, co najmniej 3 znaki, po utworzeniu obiektu nie może zostać zmodyfikowana, nawet wewnątrz klasy (*read only*)
- `Balance` - `decimal`, aktualny stan środków (saldo), nigdy nie może być ujemne, zapamiętane w zaokrągleniu do 4. cyfr po przecinku
- `IsBlocked` - `bool`, informacja, ze konto jest lub nie jest zablokowane

Zachowanie:

- Wszystkie operacje arytmetyczne wykonywane z dokładnością do 4. cyfr po przecinku
- Konstruktor tworzy obiekt na podstawie nazwy klienta (obligatoryjnie) i salda początkowego (opcjonalnie). Domyślnie, saldo początkowe wynosi `0`. W przypadku podania ujemnego salda początkowego, zgłaszany jest wyjątek typu `ArgumentOutOfRangeException`. Nazwa konta musi mieć przynajmniej 3 znaki, w przeciwnym przypadku zgłaszany wyjątek `ArgumentException`. W momencie utworzenia konto jest odblokowane.
- Wpłatę można wykonać dla podanej kwoty (wartość nieujemna), o ile konto nie jest zablokowane. Jeśli saldo ulega zmianie, zwracana jest wartość `true`. Metoda `Deposit` nie zgłasza wyjątków.
- Wypłatę można wykonać dla podanej kwoty (wartość nieujemna), o ile konto nie jest zablokowane oraz na koncie jest wystarczająca ilość środków. Jeśli saldo ulega zmianie, zwracana jest wartość `true`. Metoda `Withdrawal` nie zgłasza wyjątków.
- Konto można zablokować jedynie metodą `Block()` lub odblokować jedynie metodą `Unblock()`.
- Tekstową reprezentacją konta jest napis o formacie:
  - jeśli konto nie jest zablokowane: `Account name: {Name}, balance: {Balance}`
  - jeśli konto jest zablokowane: `Account name: {Name}, balance: {Balance}, blocked`

  Saldo podawane jest w zaokrągleniu do 2. miejsc po przecinku.

Na przykład:

Test	Wynik

`/* wypłaty`
`*/`
`var account = new Account("John");`
`account.Deposit(100.00m);`
`Console.WriteLine(account.Withdrawal(10.00m));`
`Console.WriteLine(account);`
`Console.WriteLine(account.Withdrawal(100.00m));`
`Console.WriteLine(account);`
`Console.WriteLine(account.Withdrawal(0.00m));`
`Console.WriteLine(account);`
`Console.WriteLine(account.Withdrawal(-10.00m));`
`Console.WriteLine(account);`
`account.Block();`
`Console.WriteLine(account.Withdrawal(10.4999m));`
`Console.WriteLine(account);`
	`True`
	`Account name: John, balance: 90.00`
	`False`
	`Account name: John, balance: 90.00`
	`False`
	`Account name: John, balance: 90.00`
	`False`
	`Account name: John, balance: 90.00`
	`False`
	`Account name: John, balance: 90.00, blocked`

Odpowiedź: (system kar: 0 %)

Zresetuj odpowiedź

```
using System;

namespace Bank
{
    class Account : IAccount
    {
        public string Name
        {
            get;
        }

        public decimal Balance
        {
            get;
            private set;
        }

        public bool IsBlocked
        {
            get; private set;
        }


        public void Block()
        {
            IsBlocked = true;
        }
```

```
        public bool Deposit(decimal amount)
        {
            if (amount < 0)
                return false;
            else if (IsBlocked)
                return false;
            else
            {
                Balance = decimal.Round(Balance, 4) + decimal.Round(amount, 4);
                    return true;
            }
        }
        public void Unblock()
        {
            IsBlocked = false;
        }

        public bool Withdrawal(decimal amount)
        {
            if (amount <= 0)
                return false;
            else if (Balance < 0)
                return false;
            else if (IsBlocked)
                return false;
            else if(decimal.Round(amount,4) > decimal.Round(Balance,4))

                return false;
            else
            {
                Balance = decimal.Round(Balance, 4) - decimal.Round(amount, 4);
                return true;
```

	Test	Oczekiwane	Otrzymane	

✔	`/* weryfikacja, czy implementowany jest interfejs IAccount */`
	`var account = new Account("AAA", 100.00m);`
	`if ( account is Bank.IAccount )`
	`  Console.WriteLine("IAccount implemented");`
	`else`
	`  Console.WriteLine("IAccount not implemented");`
		IAccount implemented	IAccount implemented	✔

✔	`/* weryfikacja, czy Name jest read-only */`
	`Type MyType = typeof(Account);`
	`PropertyInfo propertyNameInfo = MyType.GetProperty("Name");`
	`Console.WriteLine( propertyNameInfo.CanWrite );`
		False	False	✔

✔	`/* Utworzenie konta dla dwóch poprawnie podanych argumentów */`
	`var account = new Account(" John   ", 100.23m);`
	`Console.WriteLine(account.Balance == 100.23m);`
	`Console.WriteLine(account.Name == "John");`
	`Console.WriteLine(!account.IsBlocked);`
	`Console.WriteLine(account);`
		True
		True
		True
		Account name: John, balance: 100.23
			True
			True
			True
			Account name: John, balance: 100.23
				✔

✔	`/* Utworzenie konta dla dwóch poprawnie podanych argumentów`
	`  zaokrąglenie kwoty w Balance */`
	`var account = new Account(" John   ", 100.23156m);`
	`Console.WriteLine(account.Balance == 100.2316m);`
	`Console.WriteLine(account.Name == "John");`
	`Console.WriteLine(!account.IsBlocked);`
	`Console.WriteLine(account);`
		True
		True
		True
		Account name: John, balance: 100.23
			True
			True
			True
			Account name: John, balance: 100.23
				✔

✔	`/* Utworzenie konta dla dwóch poprawnie podanych argumentów`
	`  zaokrąglenie kwoty w Balance */`
	`var account = new Account(" John   ", 100.23511m);`
	`Console.WriteLine(account.Balance == 100.2351m);`
	`Console.WriteLine(account.Name == "John");`
	`Console.WriteLine(!account.IsBlocked);`
	`Console.WriteLine(account);`
		True
		True
		True
		Account name: John, balance: 100.24
			True
			True
			True
			Account name: John, balance: 100.24
				✔

✔	`/* Utworzenie konta dla jednego poprawnie podanego argumentu */`
	`var account = new Account(" Adam ");`
	`Console.WriteLine(account.Balance == 0.0m);`
	`Console.WriteLine(account.Name == "Adam");`
	`Console.WriteLine(!account.IsBlocked);`
	`Console.WriteLine(account);`
		True
		True
		True
		Account name: Adam, balance: 0.00
			True
			True
			True
			Account name: Adam, balance: 0.00
				✔

✔	`/* Utworzenie konta dla dwóch argumentów, ujemne saldo początkowe */`
	`try`
	`{`
	`  var account = new Account("Jim", -100.01m);`
	`  Console.WriteLine(account);`
	`}`
	`catch (ArgumentOutOfRangeException)`
	`{`
	`  Console.WriteLine("negative argument");`
	`}`
		negative argument	negative argument	✔

✔	`/* Utworzenie konta dla dwóch argumentów, nazwa jest null */`
	`try`
	`{`
	`  var account = new Account(null, 100.0m);`
	`  Console.WriteLine(account);`
	`}`
	`catch (ArgumentOutOfRangeException)`
	`{`
	`  Console.WriteLine("Name is null");`
	`}`
		Name is null	Name is null	✔

✔	`/* Utworzenie konta dla dwóch argumentów,`
	`nazwa jest zbyt krótka */`
	`try`
	`{`
	`  var account = new Account("   Jo   ", 100.0m);`
	`  Console.WriteLine(account);`
	`}`
	`catch (ArgumentException)`
	`{`
	`  Console.WriteLine("Name is to short");`
	`}`
		Name is to short	Name is to short	✔

✔	`/* Utworzenie konta dla jednego argumentu,`
	`  nazwa jest null */`
	`try`
	`{`
	`  var account = new Account(null);`
	`  Console.WriteLine(account);`
	`}`
	`catch (ArgumentOutOfRangeException)`
	`{`
	`  Console.WriteLine("Name is null");`
	`}`
		Name is null	Name is null	✔

✔	`/* Utworzenie konta dla jednego argumentu,`
	`  nazwa jest za krótka */`
	`try`
	`{`
	`  var account = new Account("   An ");`
		Name is to short	Name is to short	✔

	`Console.WriteLine(account);`
	`}`
	`catch (ArgumentException)`
	`{`
	`Console.WriteLine("Name is to short");`
	`}`			

✔	`/* zablokowanie - odblokowanie konta`
	`*/`
	`var account = new Account("John");`
	`account.Block();`
	`Console.WriteLine( account.IsBlocked );`
	`account.Unblock();`
	`Console.WriteLine( account.IsBlocked );`
		`True`
		`False`
			`True`
			`False`
				✔

✔	`/* wpłata, kwota dodatnia, konto nie zablokowane,`
	`zmiana salda */`
	`var account = new Account("John");`
	`account.Deposit(100.2345m);`
	`account.Deposit(10.2345m);`
	`Console.WriteLine(account);`
		`Account name: John, balance: 110.47`	`Account name: John, balance: 110.47`	✔

✔	`/* wpłata, kwota ujemna, konto zablokowane,`
	`saldo nie zmienione */`
	`var account = new Account("John");`
	`account.Block();`
	`Console.WriteLine(account.Deposit(-100.2345m));`
		`False`	`False`	✔

✔	`/* wpłata, kwota zerowa, konto zablokowane,`
	`saldo nie zmienione */`
	`var account = new Account("John");`
	`account.Block();`
	`Console.WriteLine(account.Deposit(0.0m));`
		`False`	`False`	✔

✔	`/* wypłaty`
	`*/`
	`var account = new Account("John");`
	`account.Deposit(100.00m);`
	`Console.WriteLine(account.Withdrawal(10.00m));`
	`Console.WriteLine(account);`
	`Console.WriteLine(account.Withdrawal(100.00m));`
	`Console.WriteLine(account);`
	`Console.WriteLine(account.Withdrawal(0.00m));`
	`Console.WriteLine(account);`
	`Console.WriteLine(account.Withdrawal(-10.00m));`
	`Console.WriteLine(account);`
	`account.Block();`
	`Console.WriteLine(account.Withdrawal(10.4999m));`
	`Console.WriteLine(account);`
		`True`
		`Account name: John, balance: 90.00`
		`False`
		`Account name: John, balance: 90.00`
		`False`
		`Account name: John, balance: 90.00`
		`False`
		`Account name: John, balance: 90.00`
		`False`
		`Account name: John, balance: 90.00, blocked`
			`True`
			`Account name: John, balance: 90.00`
			`False`
			`Account name: John, balance: 90.00`
			`False`
			`Account name: John, balance: 90.00`
			`False`
			`Account name: John, balance: 90.00`
			`False`
			`Account name: John, balance: 90.00, blocked`
				✔

Przeszedł wszystkie testy! ✔

Poprawnie

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

Zakończ przegląd

⇥ Ćwiczenie: konto w banku (nie oceniane)

Przejdź do...

Zadanie: Konto w banku (part 2) ⇤