Pierwsza klasa — konto bankowe

Czas: 2 × 45 min | praca indywidualna | Wykład 1.13

Cel: Zaprojektujesz klasę `KontoBankowe` z hermetyzacją danych, konstruktorem, metodami wpłaty i wypłaty oraz historią operacji.

Po zajęciach potrafisz:

- definiuje klasę z polami prywatnymi, konstruktorem i metodami
- chroni stan obiektu przed niepoprawnymi zmianami (hermetyzacja)
- zgłasza wyjątek przy niepoprawnym argumencie
- nadpisuje metodę `ToString`

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- pendrive lub konto w repozytorium do zapisania projektu

Oprogramowanie

- **Visual Studio Community 2026 (obciążenie „Programowanie aplikacji klasycznych platformy .NET")** — darmowe dla uczniów i nauczycieli; w pracowni zainstalowane visualstudio.microsoft.com/pl/downloads
- **.NET 10 SDK (LTS)** — instalowany razem z Visual Studio; sprawdzenie: `dotnet --version` dotnet.microsoft.com/pl-pl/download/dotnet/10.0

Bezpieczeństwo

- Zapisuj projekt w swoim folderze (np. `C:\Users\<login>\source\repos`), a na koniec zajęć skopiuj go na pendrive lub wyślij do repozytorium.
- Nie uruchamiaj programów i pakietów z nieznanych źródeł; pakiety NuGet instaluj tylko te, które wskazuje zadanie.

Przebieg zadania

1 Projekt klasy

Utwórz projekt `Bank`. Zanim napiszesz kod, zapisz w sprawozdaniu projekt klasy: pola, konstruktor, metody.

1. Pola prywatne: `_numer`, `_wlasciciel`, `_saldo` (`decimal`), `_historia` (`List<string>`).
2. Konstruktor przyjmuje numer i właściciela; saldo początkowe 0.
3. Właściwość tylko do odczytu `public decimal Saldo => _saldo;`.
4. Sprawdź, że instrukcja `konto._saldo = 1000000;` w programie głównym się nie kompiluje — zapisz komunikat błędu.

2 Wpłata i wypłata

Metody zmieniające saldo.

```
public bool Wyplac(decimal kwota)
{
    if (kwota <= 0) throw new ArgumentOutOfRangeException(nameof(kwota));
    if (kwota > _saldo) { /* zapisz odmowę */ return false; }
    _saldo -= kwota;
    // zapisz operację
    return true;
}
```

1. `Wplac(decimal kwota)` — dla kwoty ≤ 0 zgłoś `ArgumentOutOfRangeException(nameof(kwota))`.
2. `bool Wyplac(decimal kwota)` — gdy brak środków, nie zmieniaj salda i zwróć `false`.
3. Każdą operację (także odmowę) dopisz do historii z datą: `$"{DateTime.Now:dd.MM HH:mm} wpłata {kwota,10:F2} zł"`.

3 Historia i opis konta

Udostępnij dane do odczytu.

1. Dodaj właściwość `IReadOnlyList<string> Historia => _historia;` — dlaczego nie `List<string>`?
2. Nadpisz `ToString()`, aby zwracał właściciela, numer i saldo.
3. W programie głównym utwórz dwa konta i wykonaj na nich po kilka operacji, w tym nieudaną wypłatę i wpłatę ujemną w bloku `try`.

4 Przelew (rozszerzenie)

Dodaj metodę `bool Przelej(KontoBankowe cel, decimal kwota)`.

1. Przelew to wypłata z jednego konta i wpłata na drugie — wykorzystaj istniejące metody.
2. Sprawdź przelew na to samo konto i przelew bez pokrycia.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ projekt klasy (pola, metody) przed kodowaniem
- ☐ kod klasy i programu
- ☐ zrzut ekranu historii operacji
- ☐ komunikat błędu przy próbie dostępu do pola prywatnego

Pytania kontrolne

1. Co to jest hermetyzacja i jak ją zrealizowano w klasie KontoBankowe?
2. Dlaczego udostępniamy historię jako `IReadOnlyList<string>`?
3. Kiedy metoda powinna zgłosić wyjątek, a kiedy zwrócić `false`?
4. Czym różni się klasa od obiektu? Odpowiedz na przykładzie z zadania.

Kryteria oceny

Kryterium	Punkty
Projekt klasy przed kodowaniem	2
Pola prywatne, konstruktor, właściwość Saldo	4
Wpłata i wypłata z kontrolą	5
Historia, ToString, program testowy	4
Przelew	2
Pytania	3
Razem	**20**

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).