Dziedziczenie — pracownicy

Czas: 2 × 45 min | praca indywidualna | Wykład 1.15

Cel: Zbudujesz hierarchię klas pracowników z abstrakcyjną klasą bazową i metodą wirtualną `ObliczPensje`, a następnie przetworzysz listę różnych pracowników jedną pętlą.

Po zajęciach potrafisz:

- tworzy klasę abstrakcyjną i klasy pochodne
- nadpisuje metody słowem `override` i wywołuje `base`
- przetwarza listę obiektów typu bazowego (polimorfizm)
- dobiera dane i metody do odpowiedniego poziomu hierarchii

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 Klasa bazowa

Utwórz projekt `Kadry`.

1. Klasa abstrakcyjna `Pracownik` z właściwościami `Imie`, `Nazwisko` ustawianymi w konstruktorze.
2. Metoda abstrakcyjna `decimal ObliczPensje()`.
3. Nadpisz `ToString()` — zwraca imię i nazwisko.
4. Sprawdź, że `new Pracownik(...)` się nie kompiluje. Zapisz komunikat.

2 Klasy pochodne

Utwórz trzy klasy dziedziczące po `Pracownik`.

```
abstract class Pracownik(string imie, string nazwisko)
{
    public string Imie { get; } = imie;
    public string Nazwisko { get; } = nazwisko;
    public abstract decimal ObliczPensje();
    public override string ToString() => $"{Imie} {Nazwisko}";
}

class Stazysta(string imie, string nazwisko, decimal stypendium)
    : Pracownik(imie, nazwisko)
{
    public override decimal ObliczPensje() => stypendium;
}
```

1. `Etatowy`: pensja zasadnicza + premia procentowa.
2. `Godzinowy`: stawka × godziny; godziny ponad 160 płatne 150%.
3. `Stazysta`: stałe stypendium; `ToString()` dopisuje „(staż)" z użyciem `base.ToString()`.
4. Obliczenia sprawdź ręcznie: Godzinowy 40 zł × 172 h = 6400 + 12 × 60 = 7120 zł.

3 Lista płac

Polimorfizm w praktyce.

1. Utwórz `List<Pracownik>` z co najmniej 5 pracownikami różnych typów.
2. W jednej pętli wypisz typ (`GetType().Name`), pracownika i pensję w kolumnach oraz sumę.
3. Wypisz tylko pracowników godzinowych, używając `if (p is Godzinowy g)`.
4. Narysuj w sprawozdaniu diagram klas (prostokąty i strzałki dziedziczenia).

4 Nowy typ (rozszerzenie)

Dodaj klasę `Kierownik` dziedziczącą po `Etatowy` z dodatkiem funkcyjnym.

1. Użyj `base.ObliczPensje()` w nadpisanej metodzie.
2. Sprawdź, czy pętla z kroku 3 działa bez żadnej zmiany.
3. Zatwierdź pracę w Git.

W sprawozdaniu umieść

- ☐ diagram klas
- ☐ kod klas i programu
- ☐ zrzut listy płac
- ☐ ręczne obliczenia pensji dla każdego typu

Pytania kontrolne

1. Czym różni się metoda `abstract` od `virtual`?
2. Dlaczego nie można utworzyć obiektu klasy abstrakcyjnej?
3. Co to jest polimorfizm? Wskaż go w swoim programie.
4. Do czego służy słowo `base`?

Kryteria oceny

Kryterium	Punkty
Klasa abstrakcyjna	3
Trzy klasy pochodne z poprawnymi obliczeniami	6
Lista płac i operator is	4
Diagram klas	2
Kierownik z base	2
Pytania	3
Razem	**20**

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