Sprawozdanie z laboratorium

Wyższa Szkoła Ekonomii i Informatyki w Krakowie

Ćw nr:	Temat:
L-2	Implementacja oprogramowania zgodnie z diagramami UML, w tym asocjacji, agregacji i generalizacji
Nazwisko i imię:	**Nr albumu:**
Kacper Adamczyk	15606
Grupa Lab:	**Data wykonania:**
Lab2	28.03.2025

1. Opis środowiska pracy

- Komputer nr 1
- System operacyjny: Windows 11
- Środowisko programistyczne: Visual Studio Code 1.96.2
- Platforma .NET: .NET 9.0

2. Wstęp

Celem laboratorium było zapoznanie się z implementacją oprogramowania zgodnie z diagramami UML, ze szczególnym uwzględnieniem mechanizmów dziedziczenia, który jest kluczowym elementem programowania obiektowego.

W ramach zajęć przeanalizowaliśmy dwie klasy bazowe - `Pojazd` oraz `Pracownik`, wraz z ich klasami pochodnymi, które reprezentują typowe wzorce i ogólny zarys tematu.

3. Cel i zakres pracy

Głównym celem laboratorium było:

1. Praktyczne zrozumienie koncepcji dziedziczenia i polimorfizmu
2. Implementacja klas zgodnie z diagramami UML
3. Rozszerzenie klasy `Pracownik` o metody obliczania wynagrodzenia dziennego, miesięcznego i rocznego
4. Zastosowanie metod wirtualnych, abstrakcyjnych i ich nadpisywania

W ramach laboratorium zrealizowałem:

1. Analizę istniejących klas `Pojazd` i `Pracownik`
2. Implementację metod obliczania wynagrodzeń dla różnych typów pracowników
3. Dodanie specyficznej funkcjonalności dla pracownika zdalnego (bonus za terminowość)

4. Metodologia

W trakcie laboratorium najpierw przeanalizowałem strukturę istniejących klas, aby zrozumieć hierarchię dziedziczenia. Następnie zidentyfikowałem wspólne cechy klas pracowników, co pomogło w zaprojektowaniu metod obliczania wynagrodzeń w klasie bazowej. Po zaimplementowaniu podstawowych wersji metod, rozszerzyłem ich funkcjonalność w klasach pochodnych wykorzystując polimorfizm. Na końcu przetestowałem działanie wszystkich zaimplementowanych metod.

5. Opis zadań

5.1. Analiza klasy Pojazd

Klasa `Pojazd` [1] jest klasą abstrakcyjną, która definiuje podstawowe właściwości i zachowania każdego pojazdu. Na podstawie tej klasy utworzone zostały klasy pochodne: `SamochodOsobowy`, `Motocykl` i `Ciezarowka`.

Klasa bazowa zawiera właściwości(`Marka`, `TypSilnika` ...)
Oraz metody:

- Abstrakcyjna metoda `WyswietlInformacje()`, która musi być zaimplementowana przez klasy pochodne
- Wirtualna metoda `KosztPrzejazduNa100km()`, która może być nadpisana przez klasy pochodne

5.2. Analiza klasy Pracownik

Klasa `Pracownik` [2] również jest klasą abstrakcyjną, która definiuje cechy i zachowania pracowników. Na jej podstawie utworzone zostały klasy pochodne: `PracownikBiurowy`, `Menedzer` i `PracownikZdalny`.

5.3. Implementacja metod obliczania wynagrodzeń

Głównym zadaniem było rozszerzenie klasy `Pracownik` i jej pochodnych o metody obliczające różne formy wynagrodzeń. W klasie bazowej zaimplementowałem metody do obliczania wynagrodzenia dziennego, miesięcznego i rocznego [3]. W przypadku wynagrodzenia dziennego przyjąłem założenie o 22 dniach roboczych w miesiącu.

Dla klasy `PracownikBiurowy` wynagrodzenie dzienne obliczane jest na podstawie stawki godzinowej i 8-godzinnego dnia pracy, a w wynagrodzeniu miesięcznym uwzględniłem dodatek za nadgodziny, gdy liczba przepracowanych godzin przekracza 160.[4]

Dla klasy `Menedzer` Wynagrodzenie dzienne i miesięczne jest standardowe, natomiast do wynagrodzenia rocznego dodawany jest bonus.[5]

5.4. Rozszerzenie klasy PracownikZdalny

Dla pracownika zdalnego dodałem dodatkową właściwość `Terminowosc`, która wpływa na jego wynagrodzenie [6]. Pracownik zdalny, który dotrzymuje terminów, otrzymuje bonusy na każdym poziomie wynagrodzenia: 10% do dziennego, 15% do miesięcznego oraz dodatkową "trzynastkę" przy rocznym [7].

6. Analiza wyników

Zaimplementowane metody pozwalają na obliczanie wynagrodzeń dla różnych typów pracowników z uwzględnieniem ich specyfiki:

1. Pracownik biurowy otrzymuje wynagrodzenie dzienne na podstawie stawki godzinowej oraz dodatkowy bonus za nadgodziny przy obliczaniu wynagrodzenia miesięcznego.
2. Menedżer ma standardowe wynagrodzenie dzienne i miesięczne, ale do rocznego doliczany jest bonus roczny.
3. Pracownik zdalny otrzymuje bonusy za terminowość na każdym poziomie wynagrodzenia.

7. Wnioski

Laboratorium pozwoliło mi praktycznie zastosować i lepiej zrozumieć kilka ważnych koncepcji programowania obiektowego:

1. **Dziedziczenie** - tworzenie hierarchii klas, które dziedziczą cechy i zachowania po klasie bazowej.
2. **Polimorfizm** - nadpisywanie metod klasy bazowej w klasach pochodnych, co pozwala na różne implementacje tego samego zachowania.
3. **Abstrakcja** - wykorzystanie klas abstrakcyjnych do definiowania wspólnego interfejsu dla klas pochodnych.

8. Odnośniki

[1] Klasa Pojazd i jej pochodne:

```
public abstract class Pojazd
{
    public string Marka { get; set; }
    public string TypSilnika { get; set; } // np. benzyna, diesel, elektryczny
    public int MaksymalnaPrędkość { get; set; }
    public double Spalanie { get; set; } // litry na 100 km
    public double Ciezar { get; set; } // w kg
    public decimal Cena { get; set; } // Cena pojazdu

    public abstract void WyswietlInformacje();
    public virtual decimal KosztPrzejazduNa100km(decimal cenaPaliwa)
    {
        return (decimal)Spalanie * cenaPaliwa;
    }
}
```

[2] Klasa Pracownik i jej pochodne:

```
public abstract class Pracownik
{
    public string Imie { get; set; }
    public string Nazwisko { get; set; }
    public string Stanowisko { get; set; }
    public decimal Wynagrodzenie { get; set; }

    public abstract void PokazInformacje();
    public virtual decimal ObliczDzienneWynagrodzenie()
    {
        return Wynagrodzenie / 22;
    }

    public virtual decimal ObliczMiesieczneWynagrodzenie()
    {
        return Wynagrodzenie;
    }
    public virtual decimal ObliczRoczneWynagrodzenie()
    {
        return Wynagrodzenie * 12;
    }
}
```

[3] Implementacja metod wynagrodzenia w klasie bazowej Pracownik:

```
public virtual decimal ObliczDzienneWynagrodzenie()
{
    return Wynagrodzenie / 22;
}

public virtual decimal ObliczMiesieczneWynagrodzenie()
{
    return Wynagrodzenie;
}

public virtual decimal ObliczRoczneWynagrodzenie()
{
    return Wynagrodzenie * 12;
}
```

[4] Implementacja metod wynagrodzenia dla klasy PracownikBiurowy:

```
public override decimal ObliczDzienneWynagrodzenie()
{
    decimal stawkaGodzinowa = Wynagrodzenie / IloscGodzinPracy;
    return stawkaGodzinowa * 8;
}

public override decimal ObliczMiesieczneWynagrodzenie()
{
    decimal wynagrodzeniePodstawowe = Wynagrodzenie;
    decimal bonusZaNadgodziny = (IloscGodzinPracy > 160) ? (IloscGodzinPracy - 160) * (Wynagrodz
    return wynagrodzeniePodstawowe + bonusZaNadgodziny;
}

public override decimal ObliczRoczneWynagrodzenie()
{
    return ObliczMiesieczneWynagrodzenie() * 12;
}
```

[5] Implementacja metod wynagrodzenia dla klasy Menedzer:

```
public override decimal ObliczDzienneWynagrodzenie()
{
    return Wynagrodzenie / 22;
}

public override decimal ObliczMiesieczneWynagrodzenie()
{
    return Wynagrodzenie;
}

public override decimal ObliczRoczneWynagrodzenie()
{
    return ObliczMiesieczneWynagrodzenie() * 12 + BonusRoczny;
}
```

[6] Deklaracja klasy PracownikZdalny z właściwością Terminowosc:

```
public class PracownikZdalny : Pracownik
{
    public int IloscDniZdalnych { get; set; }
    public bool Terminowosc { get; set; }

    // Metody wyświetlania i obliczania wynagrodzeń...
}
```

[7] Implementacja metod wynagrodzenia dla klasy PracownikZdalny:

```
public override decimal ObliczDzienneWynagrodzenie()
{
    decimal podstawowaDniowka = Wynagrodzenie / IloscDniZdalnych;
    return Terminowosc ? podstawowaDniowka * 1.1m : podstawowaDniowka; //bonus 10%
}

public override decimal ObliczMiesieczneWynagrodzenie()
{
    decimal podstawoweWynagrodzenie = Wynagrodzenie;
    return Terminowosc ? podstawoweWynagrodzenie * 1.15m : podstawoweWynagrodzenie; //bonus 15%
}

public override decimal ObliczRoczneWynagrodzenie()
{
    decimal podstawoweRoczne = ObliczMiesieczneWynagrodzenie() * 12;
    return Terminowosc ? podstawoweRoczne + Wynagrodzenie : podstawoweRoczne;
}
```