|  |
| --- |
| Uniwersytet WSB Merito |
| Programowanie Obiektowe |
| Ćwiczenia 4 - zadania |

|  |
| --- |
| Autor: Lesław Pawlaczyk  2025/06/15 |

Spis treści

[Rozdział 1 – Zadania do zrealizowania 2](#_Toc135262555)

[Zadanie Nr 1 2](#_Toc135262556)

[Zadanie Nr 2 3](#_Toc135262557)

[Zadanie Nr 3 4](#_Toc135262558)

[Zadanie Nr 4 5](#_Toc135262559)

[Zadanie Nr 5 6](#_Toc135262560)

[Zadanie nr 6 6](#_Toc135262561)

[Zadanie nr 7 7](#_Toc135262562)

[Rozdział 2 – Ocenianie 9](#_Toc135262563)

[Bibliografia 11](#_Toc135262564)

# Rozdział 1 – Zadania do zrealizowania

Wybierz jedno z poniższych zadań.

Założenia:

1. Zadanie powinno używać GUI konsolkowego podobnego do tego, które było we wcześniejszych ćwiczeniach.
2. Dane powinny być zapisywane do pliku JSON.
3. Należy używać wstrzykiwania zależności.
4. Należy używać architektury serwisowej.

## Zadanie Nr 1

**Tytuł projektu:** Prosty system zarządzania sklepem

**Opis:**

Twoim zadaniem jest stworzenie prostej konsolowej aplikacji do zarządzania sklepem. Aplikacja powinna pozwalać na dodawanie produktów do sklepu, wyświetlanie listy dostępnych produktów oraz sprzedaż produktów do klientów.

**Szczegółowe wymagania:**

1. Utwórz klasę **Product** z właściwościami: **Name**, **Price**, **Quantity**.
2. Utwórz klasę **Shop** z listą produktów oraz metodami pozwalającymi na dodawanie produktów do listy, wyświetlanie dostępnych produktów oraz sprzedaż produktów.
3. Utwórz klasę **Customer** z metodą **Buy**, która umożliwia klientowi zakup produktu ze sklepu.
4. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.
5. Zastosuj zasadę dziedziczenia - utwórz klasę **DiscountProduct**, która dziedziczy po klasie **Product** i dodaje nową właściwość **Discount**.
6. Zastosuj zasadę polimorfizmu - metoda **Buy** w klasie **Customer** powinna inaczej obsługiwać zwykłe produkty i produkty ze zniżką.

**Podpowiedzi:**

1. Klasa **Shop** może przechowywać listę produktów w strukturze danych **List<Product>**.
2. Metoda **AddProduct** w klasie **Shop** powinna przyjmować argument typu **Product** i dodawać go do listy produktów.
3. Metoda **ShowProducts** w klasie **Shop** powinna przechodzić przez listę produktów i wyświetlać informacje o każdym produkcie.
4. Metoda **SellProduct** w klasie **Shop** powinna przyjmować argumenty: produkt do sprzedaży i klient, który chce kupić produkt. Powinna ona zmniejszać ilość dostępnego produktu i wyświetlać informację o sprzedaży.
5. Metoda **Buy** w klasie **Customer** powinna przyjmować argumenty: sklep, z którego klient chce kupić produkt, i produkt, który chce kupić.

**Pamiętaj:**

Program powinien być napisany w sposób modułowy, zgodnie z zasadami SOLID, aby ułatwić ewentualne przyszłe modyfikacje. Kod powinien być również dobrze skomentowany i łatwy do zrozumienia.

## Zadanie Nr 2

**Tytuł projektu:** Prosty system zarządzania kontami bankowymi

**Opis:**

Twoim zadaniem jest stworzenie prostej konsolowej aplikacji do zarządzania kontami bankowymi. Aplikacja powinna pozwolić na tworzenie kont, wpłacanie i wypłacanie środków oraz wyświetlanie salda.

**Szczegółowe wymagania:**

1. Utwórz klasę **BankAccount** z właściwościami: **AccountNumber**, **Balance** oraz metodami **Deposit**, **Withdraw** i **DisplayBalance**.
2. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.
3. Zastosuj zasadę dziedziczenia - utwórz klasę **SavingsAccount**, która dziedziczy po klasie **BankAccount** i dodaje nową właściwość **InterestRate** oraz metodę **AddInterest**.
4. Zastosuj zasadę polimorfizmu - metoda **DisplayBalance** powinna inaczej wyświetlać salda dla zwykłych kont i kont oszczędnościowych.

**Podpowiedzi:**

1. Metoda **Deposit** powinna przyjmować kwotę jako argument i dodawać ją do salda konta.
2. Metoda **Withdraw** powinna przyjmować kwotę jako argument, sprawdzić, czy na koncie jest wystarczająco dużo środków, a następnie odjąć tę kwotę od salda konta.
3. Metoda **DisplayBalance** powinna wyświetlać aktualne saldo konta.
4. Metoda **AddInterest** w klasie **SavingsAccount** powinna obliczyć odsetki na podstawie salda i stopy procentowej, a następnie dodać te odsetki do salda konta.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

## Zadanie Nr 3

**Tytuł projektu:** Prosty system zarządzania e-sklepem

**Opis:**

Twoim zadaniem jest stworzenie prostej aplikacji do zarządzania sklepem online. Aplikacja powinna pozwalać na dodawanie produktów do koszyka, wyświetlanie zawartości koszyka i składanie zamówień.

**Szczegółowe wymagania:**

1. Utwórz klasę **Product** z właściwościami: **ProductId**, **Name**, **Price**.
2. Utwórz klasę **Cart** z listą produktów oraz metodami pozwalającymi na dodawanie produktów do koszyka, wyświetlanie zawartości koszyka oraz obliczanie całkowitej wartości produktów w koszyku.
3. Utwórz klasę **Customer** z właściwościami: **CustomerId**, **Name**, **Cart** oraz metodą **PlaceOrder**.
4. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.
5. Zastosuj zasadę dziedziczenia - utwórz klasę **DiscountProduct**, która dziedziczy po klasie **Product** i dodaje nową właściwość **Discount**.

**Podpowiedzi:**

1. Metoda **AddProduct** w klasie **Cart** powinna przyjmować argument typu **Product** i dodawać go do listy produktów.
2. Metoda **DisplayCart** w klasie **Cart** powinna przechodzić przez listę produktów i wyświetlać informacje o każdym produkcie.
3. Metoda **CalculateTotal** w klasie **Cart** powinna przechodzić przez listę produktów, obliczać sumę ich wartości i zwracać tę sumę.
4. Metoda **PlaceOrder** w klasie **Customer** powinna wywoływać metodę **CalculateTotal** dla koszyka klienta i wyświetlać informację o złożonym zamówieniu.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

## Zadanie Nr 4

**Tytuł projektu:** Prosty system rezerwacji biletów na kino

**Opis:**

Twoim zadaniem jest stworzenie prostej konsolowej aplikacji do rezerwacji miejsc na seanse filmowe w kinie. Aplikacja powinna pozwalać na wybór filmu, wybór godziny seansu, wybór miejsca oraz potwierdzenie rezerwacji.

**Szczegółowe wymagania:**

1. Utwórz klasę **Movie** z właściwościami: **Title**, **Duration**, **AgeRestriction**.
2. Utwórz klasę **Screening** z właściwościami: **Movie**, **DateTime** oraz metodą do wyświetlania dostępnych miejsc.
3. Utwórz klasę **Seat** z właściwościami: **Row**, **Number**, **IsAvailable**.
4. Utwórz klasę **Reservation** z właściwościami: **Screening**, **Seat**, **CustomerName**.
5. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.
6. Zastosuj zasadę dziedziczenia - utwórz klasę **VIPSeat**, która dziedziczy po klasie **Seat** i dodaje nową właściwość **PriceMultiplier**.

**Podpowiedzi:**

1. Metoda **DisplayAvailableSeats** w klasie **Screening** powinna przechodzić przez listę miejsc i wyświetlać tylko te, które są dostępne.
2. Po utworzeniu obiektu klasy **Reservation**, miejsce, na które została złożona rezerwacja, powinno zmienić status **IsAvailable** na **false**.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

## Zadanie Nr 5

**Tytuł projektu:** System zarządzania biblioteką

**Opis:**

Twoim zadaniem jest stworzenie prostej aplikacji do zarządzania biblioteką. Aplikacja powinna pozwalać na dodawanie i usuwanie książek, wypożyczanie książek czytelnikom i zwracanie książek do biblioteki.

**Szczegółowe wymagania:**

1. Utwórz klasę **Book** z właściwościami: **Title**, **Author**, **ISBN**, **IsAvailable**.
2. Utwórz klasę **Reader** z właściwościami: **Name**, **BorrowedBooks** (lista wypożyczonych książek).
3. Utwórz klasę **Library** z listą książek oraz metodami do dodawania książek, usuwania książek, wypożyczania książek czytelnikom i zwracania książek.
4. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.

**Podpowiedzi:**

1. Metoda **AddBook** w klasie **Library** powinna przyjmować argument typu **Book** i dodawać go do listy książek.
2. Metoda **RemoveBook** w klasie **Library** powinna przyjmować **ISBN** książki jako argument, wyszukać książkę w liście i usunąć ją.
3. Metoda **LendBook** w klasie **Library** powinna przyjmować **ISBN** książki i czytelnika jako argumenty, sprawdzić, czy książka jest dostępna, a jeśli tak, dodać książkę do listy wypożyczonych książek czytelnika i zmienić status książki na niedostępny.
4. Metoda **ReturnBook** w klasie **Library** powinna przyjmować **ISBN** książki i czytelnika jako argumenty, usunąć książkę z listy wypożyczonych książek czytelnika i zmienić status książki na dostępny.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

## Zadanie nr 6

**Tytuł projektu:** System zarządzania hotelem

**Opis:**

Twoim zadaniem jest stworzenie prostej aplikacji do zarządzania hotelem. Aplikacja powinna pozwalać na rezerwowanie pokoi, sprawdzanie dostępności oraz anulowanie rezerwacji.

**Szczegółowe wymagania:**

1. Utwórz klasę **Room** z właściwościami: **RoomNumber**, **Capacity**, **IsAvailable**, **PricePerNight**.
2. Utwórz klasę **Reservation** z właściwościami: **GuestName**, **Room**, **CheckInDate**, **CheckOutDate**.
3. Utwórz klasę **Hotel** z listą pokoi i listą rezerwacji oraz metodami do tworzenia rezerwacji, sprawdzania dostępności pokoi i anulowania rezerwacji.
4. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.

**Podpowiedzi:**

1. Metoda **MakeReservation** w klasie **Hotel** powinna przyjmować argumenty: nazwę gościa, numer pokoju, datę zameldowania i datę wymeldowania, a następnie tworzyć nową rezerwację i dodawać ją do listy rezerwacji. Pamiętaj, aby sprawdzić, czy pokój jest dostępny przed utworzeniem rezerwacji.
2. Metoda **CheckAvailability** w klasie **Hotel** powinna przyjmować datę zameldowania i datę wymeldowania jako argumenty, a następnie przechodzić przez listę pokoi i zwracać tylko te, które są dostępne w podanym okresie.
3. Metoda **CancelReservation** w klasie **Hotel** powinna przyjmować numer rezerwacji jako argument, znajdować tę rezerwację na liście, usuwać ją i zmieniać status pokoju na dostępny.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

## Zadanie nr 7

**Tytuł projektu:** System zarządzania pracownikami

**Opis:**

Twoim zadaniem jest stworzenie prostej aplikacji do zarządzania pracownikami w firmie. Aplikacja powinna pozwalać na dodawanie, usuwanie i modyfikowanie danych pracowników, a także obliczanie wynagrodzeń.

**Szczegółowe wymagania:**

1. Utwórz klasę **Employee** z właściwościami: **EmployeeId**, **Name**, **Position**, **HourlyRate**.
2. Utwórz klasę **Company** z listą pracowników oraz metodami do dodawania pracowników, usuwania pracowników, modyfikowania danych pracowników oraz obliczania wynagrodzenia dla danego pracownika.
3. Zastosuj zasadę enkapsulacji - pola klasy powinny być prywatne, a dostęp do nich powinien być zapewniony poprzez publiczne metody get i set.
4. Zastosuj zasadę dziedziczenia - utwórz klasę **Manager**, która dziedziczy po klasie **Employee** i dodaje nową właściwość **Bonus**.

**Podpowiedzi:**

1. Metoda **AddEmployee** w klasie **Company** powinna przyjmować argument typu **Employee** i dodawać go do listy pracowników.
2. Metoda **RemoveEmployee** w klasie **Company** powinna przyjmować **EmployeeId** jako argument, wyszukać pracownika o tym Id w liście i usunąć go.
3. Metoda **ModifyEmployee** w klasie **Company** powinna przyjmować **EmployeeId** i nowe dane pracownika jako argumenty, wyszukać pracownika o tym Id w liście i zaktualizować jego dane.
4. Metoda **CalculateSalary** w klasie **Company** powinna przyjmować **EmployeeId** i liczbę godzin pracy jako argumenty, wyszukać pracownika o tym Id w liście i obliczyć jego wynagrodzenie na podstawie stawki godzinowej i liczby godzin pracy. Dla menedżerów powinno także dodać premię.

**Pamiętaj:**

Kod powinien być napisany w sposób zrozumiały, dobrze skomentowany i zgodny z dobrymi praktykami programowania obiektowego.

# Rozdział 2 – Ocenianie

**Rozwiązania powinny być indywidualne**.

1. Wykonać jedno z siedmiu zadań na ocenę co najmniej dst.
2. Wynik w postaci linku do repozytorium na GitHub oraz sprawozdanie w PDF. Sprawozdanie w PDF powinno się nazywać: ProgramowanieObiektowe\_Zadanie4\_Rozwiazanie\_ImieNazwisko\_XXXXX.pdf, gdzie ImieNazwisko zastąpić swoim imieniem i nazwiskiem a XXXXX numerem albumu.
3. Rozwiązanie powinno być linkiem do repo na GitHub, gdzie będzie pokazane PDFy albo BMP w wysokiej rozdzielczości z diagramami i wskazaniami w nazwie, którego zadania dotyczą.

**Strona tytułowa**:

1. Imię i nazwisko.
2. Adres email.
3. Numer albumu.
4. Data.
5. Link do repo na Github.

**Sposób oceniania**:

1. Ocena 3 (dostateczny)

* Aplikacja działa i spełnia podstawowe wymagania zadania.
* Zastosowane są klasy i podstawowe metody.
* Występują błędy w implementacji logiki programu.

1. Ocena 3.5 (dostateczny plus)

* Aplikacja działa bezbłędnie i spełnia wszystkie podstawowe wymagania.
* Wszystkie klasy są prawidłowo skonstruowane, pola są prywatne, a dostęp do nich jest realizowany przez publiczne metody.
* Kod jest zrozumiały i posiada podstawowe komentarze.

1. Ocena 4 (dobry)

* Dodatkowo, aplikacja zawiera zastosowanie dziedziczenia tam, gdzie jest to wymagane.
* Kod jest dobrze sformatowany, każda metoda i klasa ma adekwatny komentarz.
* Aplikacja obsługuje podstawowe błędy użytkownika.

1. Ocena 4.5 (dobry plus):

* Dodatkowo, aplikacja zawiera zastosowanie interfejsów i/lub wzorców projektowych.
* Aplikacja obsługuje wszystkie możliwe błędy użytkownika i nie pozwala na wprowadzenie niepoprawnych danych.

1. Ocena 5 (bardzo dobry):

* Kod jest optymalny, składnia jest nie tylko poprawna, ale także efektywna.
* Aplikacja posiada funkcje rozszerzające podstawowe wymagania zadania.
* Kod jest doskonale sformatowany, wszystko jest skomentowane, łatwe do zrozumienia i utrzymania.

**Czas do wykonania**:

1. Do 2025/06/15.

**Uwaga:**

Przy zauważeniu, że kod powtarza się z kodem innego studenta rozwiązanie zostanie uznane za plagiat i ocenione na ocenę „ndst”.

# Bibliografia

1. <https://github.com/palles77/ProgrammingObjectOriented/raw/main/UmlTutorial.zip>
2. <https://github.com/palles77/ProgrammingObjectOriented/raw/main/UmlSpecification.pdf>
3. <https://uml.org>
4. <https://draw.io>
5. <https://www.microsoft.com/en-ww/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling>
6. <https://www.modelio.org/index.htm>
7. <https://www.softwaretestinghelp.com/uml-diagram-tutorial/>