Laboratorium 1: Paradygmat obiektowy

PIOTR SZUSTER, MGR. INŻ.

1. KLASY I OBIEKTY

Pojęcie klasy jest w paradygmacie programowania obiektowego rozumiane jako definicja obiektów, określająca ich tożsamość, dopuszczalny stan oraz zachowanie. Tożsamość to pojęcie odnoszące się do jednoznacznej identyfikacji obiektu i jego przynależności do danej klasy. Stan obiektu jest ustalony na podstawie wartości wszystkich elementów składowych (zmiennych) klasy. Metody definiują zachowanie obiektu. Stan, tożsamość oraz zachowanie stanowią łącznie cechy obiektów. Diagram UML przykładowe klasy jest przedstawiony na S1.

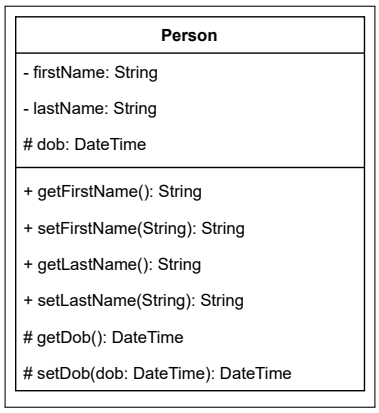

Fig. S1. Diagram klasy, reprezentującej strukturę osoby.

A. Klasa i obiekt

Na diagramach UML (Unified Modelling Language) klasy reprezentowane są przez prostokąty, składające się z trzech części. W przykładzie S1 klasa Person reprezentuje osobę. Jej nazwa określona jest wyśrodkowanym oraz pogrubionym tekstem w pierwszej części prostokąta. Druga część odpowiada za stan obiektu. Są to zmienne firstName, lastName oraz dob. Można zauważyć, że deklaracje tych zmiennych są formowane według wzorca: @nazwa: typ, gdzie @ to symbol modyfikatora dostępu. Ogólny wzorzec deklaracji zmiennej klasy jest następujący: @nazwa: typ = wart_początkowa właściwości. Metody, określające zachowanie deklaruje się podobnie do zmiennych. Deklaracja ta odbywa się w trzeciej części prostokąta. Różnica obejmuje ujęcie nazw oraz typów argumentów: @nazwa(lista_parametrów):typ_wyniku. Argumenty metody deklarowane są według wzorca: rodzajS1 nazwa : typ = wart_początkowa. Zbiór metod stanowi interfejs klasy. Obiekty są instancjami klas. Składniki statyczne klas są wyróżniane podkreśleniem. Odnoszą się one do elementów klas, a nie obiektów. Kursywą wyróżnia się metody wirtualne.

B. Hermetyzacja

Najczęściej używa się trzech modyfikatorów dostępu, które są przedstawione w tabeli S2. Modyfikatory dostępu określają widzialność pól oraz metod klasy poza jej wnętrzem. Dostęp publiczny jest umożliwiany wszystkim klasom. Dostęp chroniony jest możliwy tylko dla klas pochodnych (dziedziczących cechy). Dostęp prywatny umożliwiany jest w obrębie jedynie danej klasy. Z modyfikatorami dostępu związane jest pojęcie hermetyzacji określanej również jako enkapsulacja. Hermetyzacja wiąże dane z metodami, które na nich operują oraz ogranicza dostęp do wybranych

Table S1. Rodzaje argumentów

Rodzaj	opis
in	metoda może czytać argument, nie może zmieniać
out	metoda może zmieniać argument, nie może czytać
inout	metoda może zmieniać oraz czytać argument

Table S2. Najczęściej używane modyfikatory dostępu

Symbol	dostęp
-	prywatny
#	chroniony
+	publiczny

części obiektów. Jest wykorzystywana w celu ukrycia stanu klasy, uniemożliwiając bezpośredni dostęp do niego z zewnątrz, w sposób umożliwiający ukazanie implementacji lub naruszenie jego ciągłości. Hermetyzacja zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Gdy dostęp do wszystkich pól w klasie jest możliwy tylko i wyłącznie poprzez metody, mamy do czynienia z hermetyzacją pełną.

2. RELACJE MIĘDZY KLASAMI

W przypadku występowania wielu klas (a więc w znakomitej większości przypadków) w obrębie projektu, między klasami mogą zaistnieć relacje:

Relacje między kasami (kolejność zgodnie z ich siłą)

Dziedziczenie
Kompozycja (agregacja całkowita)
Agregacja częściowa
Asocjacja
Zależność

A. Dziedziczenie

Dziedziczenie jest relacją przekazywania cech. Dziedziczeniu podlegają wszystkie cechy. Klasa dziedzicząca nazywana jest klasą pochodną. Klasa bazowa jest uogólnieniem klasy pochodnej. Przykład dziedziczenia jest ukazany na S2. Przykład S2 definiuje klasę Person jako ogólny przypadek osoby. Klasy Student oraz Lecturer stanowią szczególne przypadki osób. Mogą korzystać z wszystkich metod klasy osoba oraz definiują własne. Relacje dziedziczenia oznacza się strzałką z grotem w kształcie trójkąta równoramiennego. Od strony trójkąta strzałki umieszcza się klasę bazową. Od strony końca linii klasę pochodną. W przypadku klasy Lecturer opcjonalnie umieszczono słowo Extends aby podkreślić, że klasa Lecturer rozszerza klasę Person.

B. Kompozycja

Kompozycja jest relacją całość-część, w której wszystkie części są tworzone i zarządzane przez całość. Całość oraz części nie mogą istnieć bez siebie - czasy ich istnienia są identyczne. W momencie usunięcia całości wszystkie części również przestają istnieć. Kompozycja jest zaznaczana linią zakończoną wypełnionym rombem. Romb umieszcza się przy całości, koniec linii przy części. Przykład kompozycji jest ukazany na S3. W przykładzie całkowita populacja ludzi zawiera wszystkie n osób.

C. Agregacja częściowa

Agregacja częściowa jest relacją całość-część, w której części nie są zarządzane przez całość. Oznacza się ją linią zakończoną pustym rombem. Romb umieszcza się przy całości, koniec linii przy części. Przykład agregacji jest ukazany na S4. W przykładzie całkowita populacja ludzi

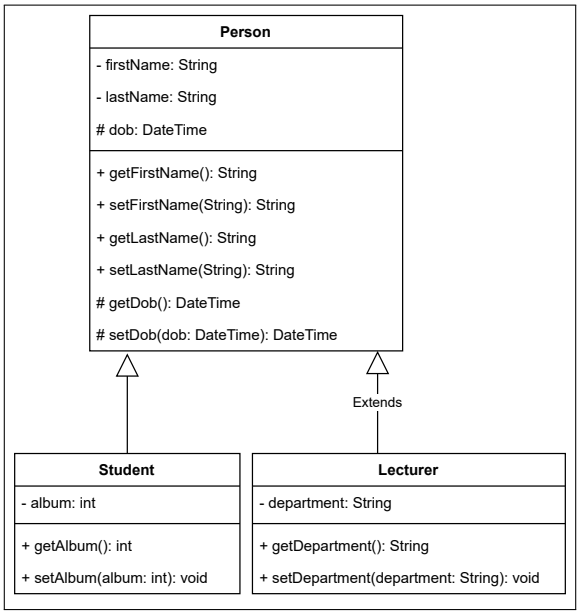

Fig. S2. Diagram klas, reprezentujący dziedziczenie.

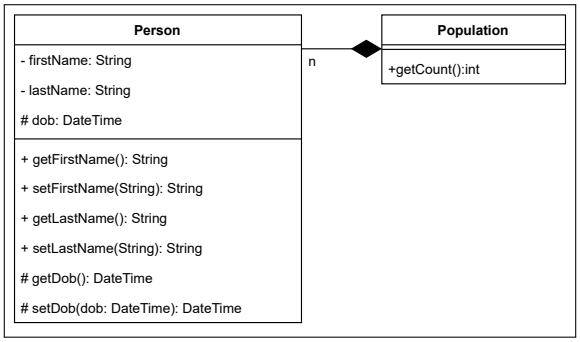

Fig. S3. Diagram klas, reprezentujący kompozycję.

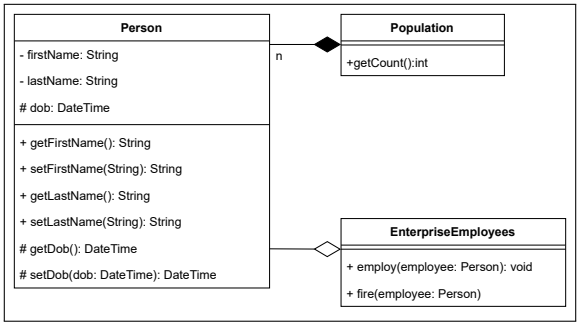

Fig. S4. Diagram klas, reprezentujący agregację oraz kompozycję.

zawiera wszystkie n osób. Część z tych osób może zostać zatrudniona w przedsiębiorstwie. W momencie likwidacji przedsiębiorstwa bądź zwolnienia osoby dalej istnieją.

D. Asocjacja

Asocjacja reprezentuje czasowe powiązanie pomiędzy obiektami dwóch klas. Obiekty związane asocjacją są od siebie niezależne. W przypadku asocjacji żaden obiekt nie jest właścicielem drugiego: nie tworzy go, nie zarządza nim, a moment usunięcia drugiego obiektu nie jest z nim związany. Asocjacje zwykle posiadają nazwy w postaci czasownika, który pozwala określić jej znaczenie w języku naturalnym. Przykład asocjacji jest ukazany na S5. W przykładzie osoby

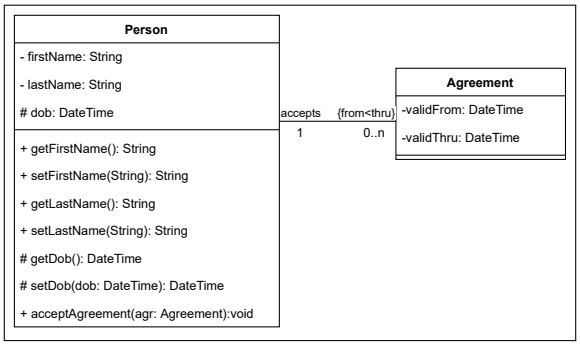

Fig. S5. Diagram klas, reprezentujący asocjację.

zawierają umowy ważne w terminach od do. Asocjacja jest obrazowana przez linię prostą.

E. Zależność

Zależność jest relacją między klasami, oznaczającą, że jedna klasa oddziaływuje na inną. W przykładzie Osoba tworzy Konto. Przykład zależności jest ukazany na S6.

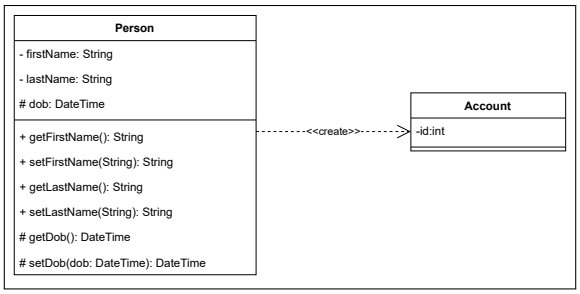

Fig. S6. Diagram klas, reprezentujący zależność

3. SOLID

SOLID to pięć podstawowych zasad programowania obiektowego.

Single responsibility principle

Każda klasa odpowiada za jedną funkcjonalność.

Open/closed principle

Każda klasa jest otwarta na rozbudowę, zamknięta na modyfikację.

Liskov substitution principle

W miejscu klasy bazowej można użyć dowolnej klasy pochodnej.

Interface segregation principle

Interfejsy powinny być konkretne i jak najmniejsze.

Dependency inversion principle

Wszystkie zależności powinny w jak największym stopniu zależeć od abstrakcji a nie od konkretnego typu.

4. POJĘCIA DODATKOWE

Polimorfizm to mechanizm umożliwiający korzystanie z jednego interfejsu encjom o różnych typach. Klasa abstrakcyjna jest to klasa, której nie można instancjonować.

Interfejs

Interfejs - klasyfikator zawierający deklaracje właściwości i metod ale nie implementujący ich. Interfejs deklaruje grupę operacji bez podawania ich implementacji. Przykład interfejsu ukazuje S7. W przykładzie S7 Kolekcja implementuje interfejs Sortowalny.

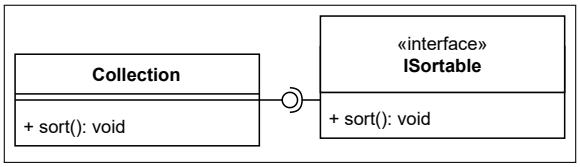

Fig. S7. Diagram klas, reprezentujący interfejs.

5. SINGLETON

Singleton jest kreacyjnym wzorcem projektowym. Ogranicza możliwość tworzenia instancji obiektów klasy do jednego egzemplarza. Klasa Singleton posiada statyczne pole singleton, prywatny konstruktor oraz statyczną metodę getInstance. Metoda getInstance zwraca instancję singleton jeśli istnieje, w innym przypadku wywołuje konstruktor i przypisuje powstałą instancję polu Singleton. Diagram wzorca Singleton jest przedstawiony na S8.

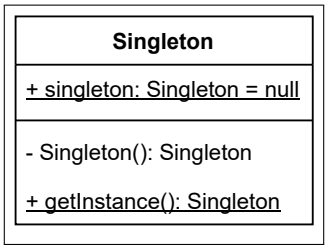

Fig. S8. Diagram kreacyjnego wzorca projektowego Singleton

6. ZADANIE LABORATORYJNE

Narodowy Bank Polski publikuje w witrynie https://www.nbp.pl/kursy/xml/lasta.xml tabelę średnich kursów walut. Zaprojektować narzędzie umożliwiające obliczenie wartości końcowej kwoty w dowolnej walucie docelowej po podaniu przez użytkownika dowolnej ilości środka pieniężnego dowolnej waluty źródłowej.

UWAGI

Dokonać dekompozycji problemu
Uwzględnić relacje między klasami w szczególności całość-część
Opracować architekturę rozwiązania
Uwzględnić zasady SOLID
Wykorzystać Singleton

A. Warunki zaliczenia zadania

Złożenie podczas zajęć laboratoryjnych projektu zawierającego

Diagram klas UML rozwiązania
Implementację rozwiązania w dowolnym obiektowym języku programowania
Zaprezentowanie zaimplementowanego rozwiązania

Rozwiązanie powinno uniemożliwiać wprowadzanie błędnych danych przez użytkownika.

Wersja: 12 października 2021, 21:37