Paradygmaty Programowania
Programowanie obiektowe

dr inż. Marcin Bączyk

Wykład 5

28 marca 2022

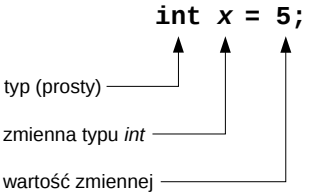
Treść dzisiejszego wykładu

- wstęp
- przykład
- mechanizmy obiektowości
- relacje między obiektami

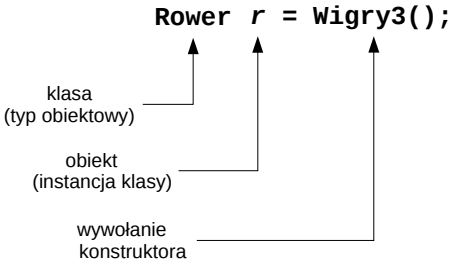
Literatura

- M. Weisfeld “Myślenie obiektowe w programowaniu”, Helion, 2010
- Robert C. Martin “Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów”, Helion, 2018
- Bartosz Walter “Zaawansowane projektowanie obiektowe”
- Grady Booch, James Rumbaugh, Ivar Jacobson “UML przewodnik użytkownika”

Zmienna typu prostego



Zmienna typu obiektowego



Programowanie obiektowe – podstawowe mechanizmy

Programowanie obiektowe jest sposobem wyrażenia programu komputerowego w postaci zbioru obiektów komunikujących się między sobą w celu wykonania zadań.

wykorzystywane mechanizmy / techniki :

- abstrakcja
- hermetyzacja
- dziedziczenie
- polimorfizm

Programowanie obiektowe – podstawowe mechanizmy

Programowanie obiektowe jest sposobem wyrażenia programu komputerowego w postaci zbioru obiektów komunikujących się między sobą w celu wykonania zadań.

wykorzystywane mechanizmy / techniki :

- abstrakcja
- hermetyzacja
- dziedziczenie
- polimorfizm
- **odpowiedzialność**

Konsekwencją stosowania podejścia czysto obiektowego jest przypisanie każdej operacji w systemie konkretnemu obiektowi i przekazanie mu odpowiedzialności za tę operację.

Programowanie obiektowe – definicje

Odpowiedzialność

1. Rodzina funkcji, które służą jednemu konkretnemu aktorowi.
2. Zakres wykonywanych czynności.
3. Zobowiązanie albo kontrakt danego typu lub klasy.

Aktor

Spójny zbiór ról odgrywanych przez użytkowników klasy lub systemu. Obiekt niebędący częścią danego systemu, który wchodzi z nim w interakcję.

Klient

Obiekt, na rzecz którego świadczona jest usługa.

Programowanie obiektowe – główne idee

- Programowanie obiektowe jest odpowiedzią na potrzebę modelowania świata rzeczywistego w możliwie wierny sposób (z dokładnością do poziomu abstrakcji).
- Systemy projektowane obiektowo składają się z dużej liczby "małych" obiektów współpracujących ze sobą.
- **Interfejs publiczny**, oznaczający usługi, jakie obiekt może świadczyć swoim klientom, informuje jedynie o tym *co* może być zrobione.
- Klienty poszczególnych obiektów nie ingerują w to, *w jaki sposób* dana usługa jest świadczona.
- Systemy obiektowe projektowane są w taki sposób aby obiekt odpowiedzialny za daną funkcjonalność mógł być zastąpiony innym *bez zmiany kodu klienta*.
- Żaden z podstawowych mechanizmów programowania obiektowego nie musi być wspierany przez dany język, aby móc napisać w nim program obiektowy.

Obiekt

Obiekt istniejący w programie często (lecz nie zawsze!) reprezentuje konkretny obiekt w świecie rzeczywistym.

Każdy obiekt posiada:

- tożsamość (np. adres w pamięci),
- stan (wartości atrybutów),
- zachowanie (metody, czyli usługi które mogą świadczyć klientom).

Obiekt jest elementem modelowanej dziedziny (pewnego obszaru rzeczywistości), odpowiadającym za wybrany jej fragment.

Od obiektu zależy w jaki sposób realizowana jest odpowiedzialność.

Przykład

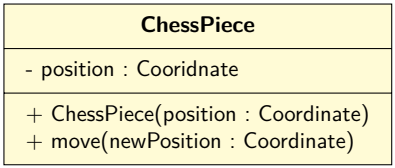
Przykład pokazujący różnice w dekompozycji obiektowej i funkcjonalnej

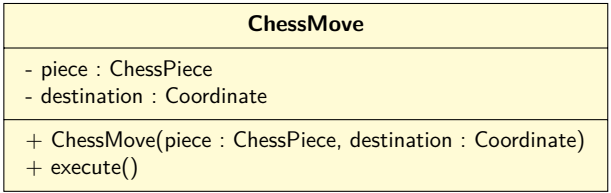
Klasa – typ

W językach obiektowych, w których występuje pojęcie **klasy**, oznacza ono sposób definiowania obiektów. Definicja klasy zawiera:

- **atrybuty** – stan obiektu klasy (zmienne, stałe),
- **metody** – możliwe zachowania obiektu (funkcje).

Obiekty utworzone na podstawie danej klasy nazywane są jej **instancjami**.
Do inicjalizacji stanu obiektów podczas ich tworzenia służy specjalna metoda nazywana **konstruktorem**.



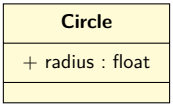


Mechanizmy obiektowości – hermetyzacja

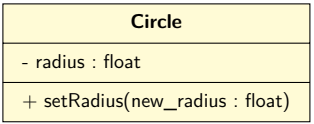
Hermetyzacja

- Ukrywanie przed klientami szczegółów implementacji obiektu.
- Podnosi stopień abstrakcji.
- Zabezpiecza obiekt przed "nieumiejętną" modyfikacją.

Bez hermetyzacji



Z hermetyzacją



Implementacja metody `setRadius`:

```
def setRadius(new_radius: float):
    if new_radius > 0:
        self.radius = new_radius
    else:
        # Zgłoś nieprawidłowe działanie
```

Mechanizmy obiektowości – polimorfizm

Polimorfizm (z gr. *poly* – wiele, *morph* – postać)

Polimorfizm jest mechanizmem umożliwiającym danemu klientowi korzystanie z różnych obiektów niezależnie od ich konkretnego typu, o ile posiadają jednakowy interfejs.

Dzięki polimorfizmowi wybór konkretnej metody wywoływanej przez klienta nie zależy od niego samego (czyt. brak instrukcji `if` / `then` / `else` / etc.), a od konkretnego rodzaju obiektu świadczącego usługi.

Polimorfizm charakteryzuje się posiadaniem przez obiekty różnych typów metod o takiej samej sygnaturze.

Rodzaje polimorfizmu

- dynamiczny – czasu wykonania
- statyczny – czasu kompilacji (jeżeli kod jest kompilowany)

Mechanizmy obiektowości – polimorfizm

Polimorfizm dynamiczny

Decyzja o tym, dokąd jest przekazywane sterowanie w programie zależy od obiektu i wyznaczana jest w czasie działania programu.

W językach kompilowanych (np. Java, C++) do polimorficznego wywoływania metod wykorzystywane jest mechanizm dziedziczenia lub implementacji interfejsu.

W językach interpretowanych (np. Matlab, Python) do polimorficznego wywołania metody mechanizm dziedziczenia oraz implementacji interfejsu nie jest niezbędny.

Polimorfizm statyczny

Decyzja o przekazaniu sterowania zależy od typu obiektu i wyznaczana jest w trakcie kompilacji programu.
W językach z rozbudowanym programowaniem generycznym (C++) ten rodzaj polimorfizmu wykorzystywany jest do tworzenia rodziny klas o podobnej strukturze.

Polimorfizm – przykład

- Jak wypisać tekst na ekranie i zapisać go do pliku?
- Podejście proceduralne:

```
display("Cokolwiek");
save_to_file("Cokolwiek");
```

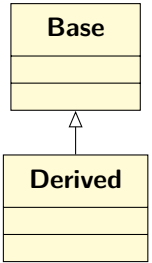
- Podejście obiektowe:

```
OutputDevice out_dev;
out_dev = Display();
out_dev.write("Cokolwiek");
out_dev = File();
out_dev.write("Cokolwiek");
```

Dziedziczenie

Dziedziczenie jest mechanizmem współdzielenia funkcjonalności między klasami.

Dzięki dziedziczeniu klasa oprócz swoich własnych atrybutów oraz zachowań uzyskuje także dostęp do atrybutów i zachowań klasy, po której dziedziczy.



Klasa, z której następuje dziedziczenie, nazywana jest klasą **bazową**.

Klasa dziedzicząca po klasie bazowej nazywana jest klasą **pochodną**.

Dygresja – interfejs

Interfejs

W programowaniu obiektowym można wyróżnić dwa znaczenia słowa interfejs. Można je rozumieć jako:

1. Zestaw publicznych atrybutów i metod obiektu, które są dostępne dla jego klientów.

   Niekiedy interfejsem publicznym określa się również wszystkie (publiczne i prywatne) atrybuty i metody klasy.

2. Abstrakcyjną klasę bazową definiującą zestaw metod, które klasy pochodne muszą zaimplementować.

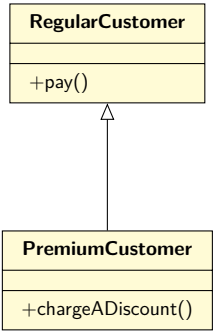
   Klasą abstrakcyjną nazywa się klasę, która nie ma swoich instancji.

Dziedziczenie a implementacja interfejsu

Dziedziczenie powoduje przeniesienie z klasy bazowej do klasy pochodnej *typu oraz implementacji*.

Implementacja (realizacja) interfejsu powoduje przeniesienie z klasy bazowej do klasy pochodnej *jedynie typu*.

Korzystanie z interfejsów skutkuje powstawaniem słabszych zależności pomiędzy klasami.

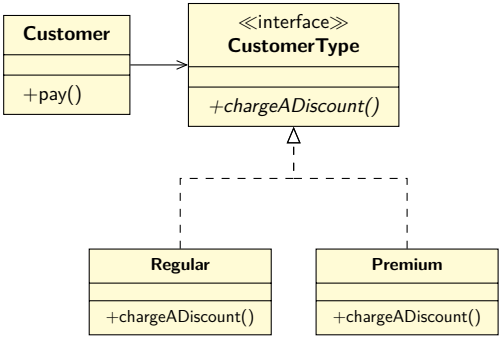


Dziedziczenie a implementacja interfejsu

Dziedziczenie powoduje przeniesienie z klasy bazowej do klasy pochodnej *typu oraz implementacji*.

Implementacja (realizacja) interfejsu powoduje przeniesienie z klasy bazowej do klasy pochodnej *jedynie typu*.

Korzystanie z interfejsów skutkuje powstawaniem słabszych zależności pomiędzy klasami.



Mechanizmy obiektowości – abstrakcja

Abstrakcja (z łac. *abstractio* – oderwanie)

Abstrakcja, w kontekście projektowania obiektowego, jest zdolnością do pomijania niektórych, nieistotnych aspektów modelowanego fragmentu rzeczywistości.

Abstrahowaniu podlegają zarówno dane przechowywane przez obiekty, jak również ich zachowanie.

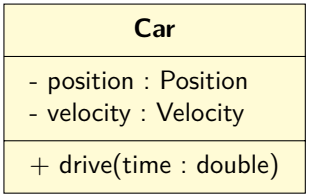
Abstrakcję można rozumieć na dwa sposoby:

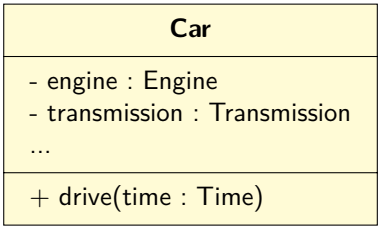
1. jako zbiór istotnych w kontekście rozwiązywanego problemu parametrów i cech zachowania danego obiektu oraz
2. jako zestaw cech wspólnych większej ilości obiektów.

Abstrakcja

W zależności od budowanego systemu, poszczególne fragmenty będą modelowane na różnym poziomie szczegółowości.

Inaczej będzie modelowany samochód na potrzeby prostej gry zręcznościowej, a inaczej złożony symulator do nauki jazdy samochodem.



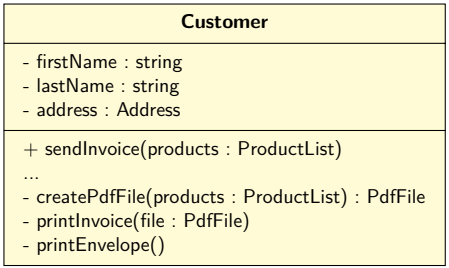


Mechanizmy obiektowości – abstrakcja

Abstrakcja

- Abstrakcja pozwala na odsunięcie w czasie konieczności podejmowania decyzji projektowych.
- Dzięki abstrakcji projektant może się skupić na istotnych cechach modelowanego problemu odsuwając (być może na zawsze) konieczność zajmowania się nieistotnymi jego aspektami.
- Pozwala na stosowanie warstw pośredniczących.
- Pozwala korzystnie wpływać na koszt utrzymania oprogramowania.
- System zbudowany z abstrakcyjnych komponentów może być łatwo rozszerzany.

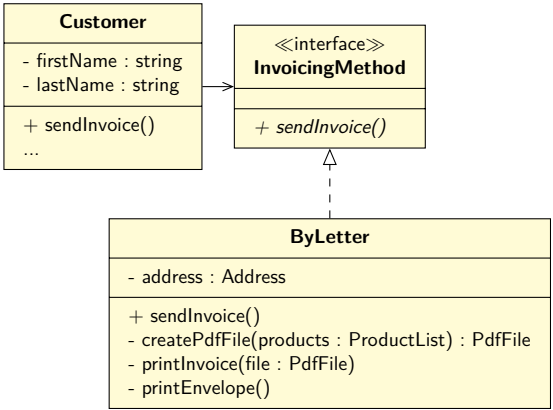
Abstrakcja – przykład



Słaba abstrakcja

Sposób wysyłki faktury "zaszyty" w kodzie (PDF, wydruk, poczta itp.)

Abstrakcja – przykład cd.



Lepsza abstrakcja

Szczegóły decyzji projektowych oderwane od implementacji klasy

Należy hermetyzować decyzje, które na późniejszym etapie projektu mogą ulec zmianie, np.:

- zmiana zachowania metody,
- sposób przechowywania danych.

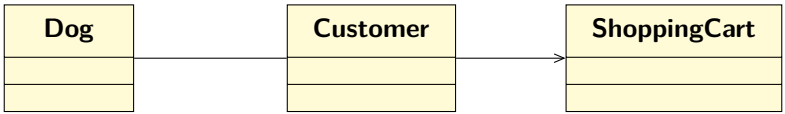
Relacje pomiędzy obiektami – asocjacja

Asocjacja – idea

- Asocjacja to relacja typu “ X używa Y”, “ X komunikuje się z Y”.
- Asocjacja może być jedno- lub dwukierunkowa (symetryczna).
- Obiekty X i Y istnieją niezależnie od siebie.
- Najsłabsza z możliwych relacji.

Asocjacja – kwestie implementacyjne

- W obiekcie X istnieje referencja (odnośnika) do obiektu Y.
- Dzięki temu X może wywoływać metody obiektu Y.
- Obiekty nie są ze sobą związane na stałe i mogą się zmieniać na inne.



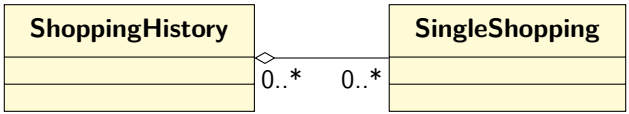
Relacje pomiędzy obiektami – agregacja

Agregacja – idea

- Agregacja to relacja typu “X posiada Y”.
- Relacja jest asymetryczna.
- Obiekty X i Y z reguły istnieją niezależnie od siebie.

Agregacja – kwestie implementacyjne

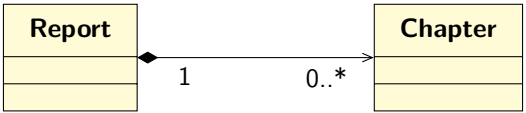
- Obiekt agregujący przechowuje referencje do obiektów agregowanych oraz zarządza nimi poprzez:
  - dodawanie i usuwanie oraz
  - przeszukiwanie w odpowiedzi na żądania swoich klientów.
- Obiekty agregowane mogą należeć do wielu obiektów agregujących.



Relacje pomiędzy obiektami – kompozycja

Kompozycja

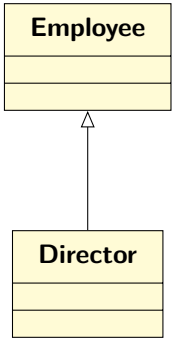
- Kompozycja to relacja typu “X składa się z obiektów klasy Y”.
- Relacja asymetryczna podobna do **agregacji**, ale silniejsza.
- Obiekty “komponowane” Y mogą należeć tylko do jednego obiektu X.
- Obiekty Y nie mają racji bytu poza obiektem X.
- Wybór między agregacją a kompozycją jest często kwestią implementacji.



Relacje pomiędzy obiektami – dziedziczenie

Dziedziczenie

- Klasa **pochodna** jest tworzona na podstawie innej – **bazowej**.
- Klasa pochodna jest zmodyfikowaną i/lub rozbudowaną wersją klasy bazowej:
  - ma dodatkowe metody (interfejs rozbudowany względem klasy bazowej) i/lub
  - ma zdefiniowane na nowo działanie pewnych metod odziedziczonych po klasie bazowej
- Dziedziczenie jest najsilniejszym rodzajem relacji.
- Możliwe naruszenie hermetyzacji



Silna zależność pomiędzy dwoma typami powoduje duży stopnień sprzężenia, utrudniającego rozwój oprogramowania. Jeśli to możliwe, dziedziczenie należy zastępować użyciem **interfejsów**.

Relacje pomiędzy obiektami – realizacja (implementacja) interfejsu

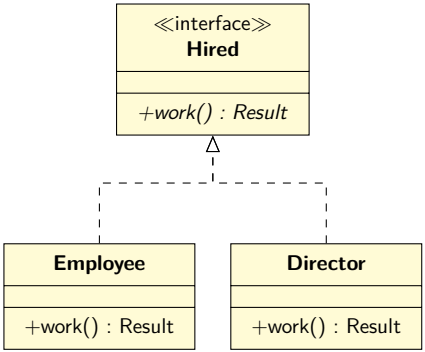
Realizacja (implementacja interfejsu)

W tym znaczeniu interfejs to klasa bazowa zawierająca *wyłącznie*:

- **sygnatury metod** (typ argumentów i wartości zwracanych) bez ich implementacji,
- ewentualnie także definicje stałych.

Realizacja jest rodzajem relacji słabszym niż dziedziczenie. Tworzona klasa:

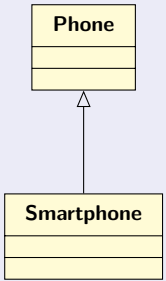
- implementuje *sygnatury* metod narzucone przez interfejs,
- nie dziedziczy żadnych atrybutów i ciał metod, *ponieważ w interfejsie nie są zdefiniowane*



Dziedziczenie a kompozycja

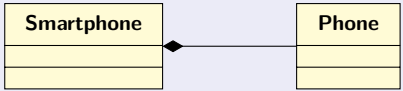
Dziedziczenie

"Smartfon to telefon, jak każdy inny, ale z dodatkowymi funkcjami".



Kompozycja

"Smartfon to urządzenie składające się z wielu modułów. Jeden z nich umożliwia prowadzenie rozmów telefonicznych".



Dziedziczenie a kompozycja cd.

Dziedziczenie

- relacja silna mogąca łamać hermetyzację
- relacja ustalana w czasie kompilacji
- klasa pochodna dziedziczy typ oraz atrybuty i zachowanie klasy bazowej

Kompozycja lub agregacja

- relacja relatywnie słaba
- ustalana w trakcie wykonywania programu
- wiąże obiekty jedynie przez typ agregowanych obiektów

Tam, gdzie to możliwe, należy preferować kompozycję nad dziedziczenie.

Relacje pomiędzy obiektami – podsumowanie

Relacje między klasami (lub obiektami klas) – od najsłabszej do najsilniejszej:

- **asocjacja** – X używa Y, X komunikuje się z Y itp.,
- **agregacja** – X posiada Y (ale nie na własność),
- **kompozycja** – X składa się z Y, X jest wyłącznym właścicielem Y,
- **implementacja** interfejsu – Y zachowuje się zgodnie z wymaganiami interfejsu X,
- **dziedziczenie** – Y jest zmodyfikowaną lub rozbudowaną wersją X.

Programowanie obiektowe – najważniejsze cechy

Podsumowanie

W programowaniu obiektowym:

- Podstawą systemu są współpracujące ze sobą obiekty.
- Obiekty mogą być powiązane ze sobą za pomocą kilku rodzajów relacji.
- Obiekty przyjmują na siebie odpowiedzialność za wybrany obszar implementowanego modelu.
- Mechanizmy obiektowe pozwalają w prawidłowy sposób projektować system i przydzielać odpowiedzialność obiektom.
- Klienci danego obiektu nie wiedzą, w jaki sposób realizowana jest jego odpowiedzialność.
- Obiekty mogą być przekazywane pomiędzy modułami bez wprowadzania dodatkowych (zbędnych) zależności.
- Rozwój (dodawanie nowych) funkcjonalności odbywa się bez zmiany wysokopoziomowych reguł biznesowych (ograniczeń wynikających z logiki i specyfiki danej dziedziny).