Modelowanie Systemu Informatycznego

prezentacja 6

Modelowanie struktury – diagram klas

wersja 1.0

dr inż. Paweł Głuchowski

Wydział Informatyki i Telekomunikacji, Politechnika Wrocławska

Treść prezentacji

1

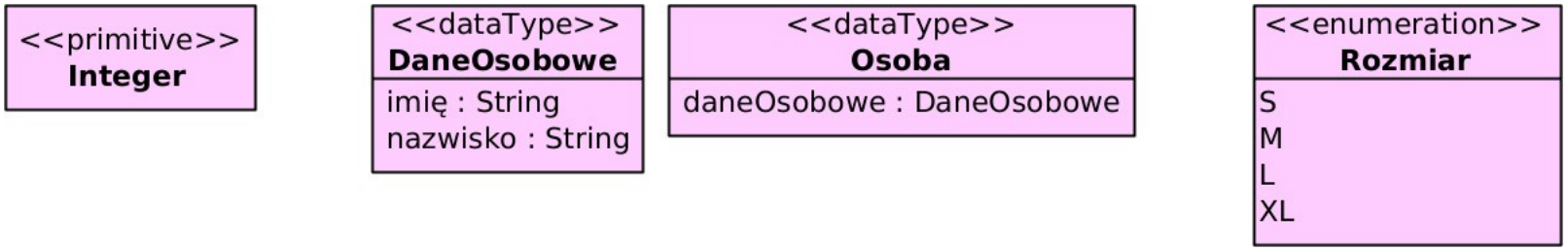
Klasyfikatory

Klasyfikator /classifier/

- **Abstrakcyjna metaklasa** instancji o wspólnych własnościach (cechach).
  - Zorganizowana hierarchicznie w związkach uogólnienia.
  - Może być szablonem w celu dalszej redefinicji.
- Klasyfikator ma nazwę i nad nią (opcjonalnie) stereotyp w nawiasach «».
- Klasyfikator ma następujące przedziały /compartment/:
  - **atrybuty** /attributes/ – własności, które go cechują;
  - **operacje** /operations/ – operacje, które wykonuje;
  - **odbiory** /receptions/ – sygnały, które odbiera;
  - inne, typowe dla danego klasyfikatora.
- **Podział klasyfikatorów:**
  - klasyfikatory proste /simple/: typ danych, sygnał, odbiór, interfejs.
  - klasyfikatory złożone /structured/ – mają złożoną strukturę wewnętrzną: klasa, asocjacja, klasyfikator opakowany, komponent, współdziałanie.

Typ danych /data type/

- Jego instancje różnią się tylko swoją wartością.
- Jest złożony, jeśli ma atrybuty.
- Ma stereotyp: «*dataType*».
- **Typ podstawowy** /primitive type/
  - Jest predefiniowany i niezłożony, np. *integer*.
  - Ma stereotyp «*primitive*».
- **Enumeracja** /enumeration/
  - Składa się z literałów (różnych wartości).
  - Ma stereotyp «*enumeration*».



Klasyfikatory

Sygnał /signal/

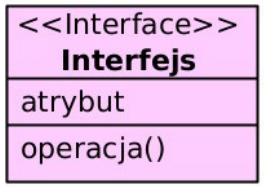
- Modeluje asynchroniczną komunikację między obiektami (powoduje reakcję obiektu, ale nie powoduje odpowiedzi).
- Ma stereotyp: «*signal*».
- Jego atrybuty to treść komunikacji.
- Może być parametryzowany /parameterized/ i ograniczony /bound/.
- Może być parametrem szablonu /template parameter/.

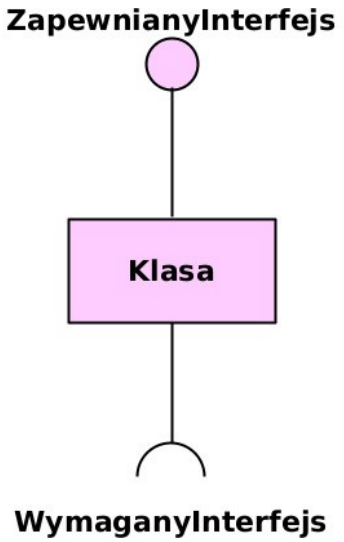
Odbiór /reception/

- Określa sygnały, na które może odpowiedzieć klasa lub interfejs.
- Znajduje się w tej klasie lub tym interfejsie w przedziale odbiorów:
  - jako operacja ze stereotypem «*signal*» i nazwą sygnału,
  - może mieć parametry wejściowe.

Interfejs /interface/

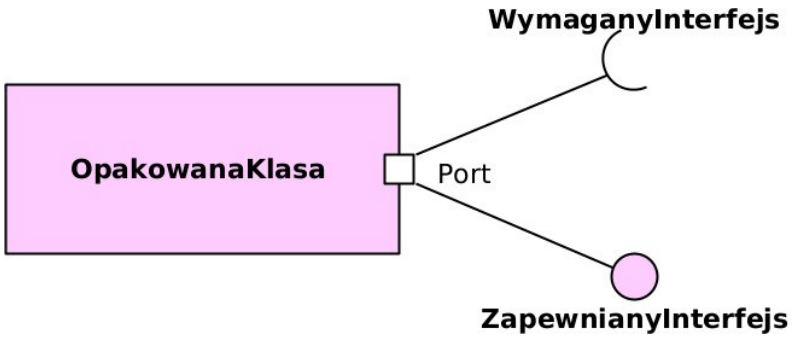
- Abstrakcja klasyfikatora (np. klasy) – jego publiczne atrybuty, operacje i wewnętrzna struktura.
  - Ogranicza zapewniającą go (implementującą) klasę (określa, co klasa ma zaimplementować).
  - NIE określa sposobu implementacji.
  - NIE ma instancji.
- Ma stereotyp: «*interface*»
  lub jest to „lizak” (gdy zapewniany) lub „łapka” (gdy wymagany).
- Operacje mogą mieć parametry.
- Może być parametryzowany /parameterized/ i ograniczony /bound/.
- Może być parametrem szablonu /template parameter/.





Opakowany klasyfikator /encapsulated classifier/

- Odizolowany od otaczającego go środowiska przez porty (1 lub więcej).
- **Port** /port/ – punkt interakcji klasyfikatora z jego otoczeniem lub jego zachowania z jego wewnętrznymi *rolami*:
  - dostarcza dostęp do klasyfikatora lub jego usługę (np. API);
  - może mieć nazwę: `<name> [:<port-type>]`.
- **Delegacja** /delegation/ – wewnętrzna relacja łącząca port z rolą klasyfikatora.



Pozostałe klasyfikatory złożone

- **Klasa** /class/ – konkretna realizacja klasyfikatora.
- **Asocjacja** /class/ – powiązanie między klasyfikatorami.
- **Komponent** /class/ – niezależny moduł systemu.
- **Współdziałanie** /class/ – powiązanie instancji klasyfikatorów w celu wykonania zadania.

2

Diagram klas

Diagram klas /class diagram/

- Modeluje obiektową, statyczną strukturę oprogramowania:
  klasy, ich stereotypy, relacje między nimi itd.
  - Instancje klas → na **diagramie obiektów**.
  - Algorytm wykonania operacji klasy, przepływ sterowania i obiektów między klasami i ich instancjami, zmiany stanów instancji klas → na **diagramach czynności**, **komunikacji**, **sekwencji**, **stanów** itd.

Diagram klas

Klasa /class/

Konkretna realizacja klasyfikatora – klasyfikuje obiekty i ich cechy.

Model bytu, powielanego jako instancje klasy (obiekty).

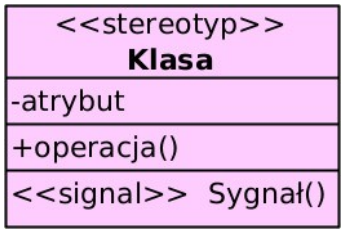
Składa się z: atrybutów, operacji, odbiorów, portów i relacji.

Klasa aktywna /active/

- Jej instancja jest aktywna:
  - posiada, wykonuje i kontroluje swój proces lub wątek;
  - procesy i wątki instancji klasy pasywnej są kontrolowane przez inne.

Zawartość klasy

- **Stereotypy** – modyfikacja klasy.
- **Nazwa** – nazwa klasy.
- Przedziały:
  - **atrybuty** /attributes/ – własności, które cechują klasę;
  - **operacje** /operations/ – operacje, które klasa wykonuje;
  - **odbiory** /receptions/ – sygnały, które klasa odbiera;
  - **wewnętrzna struktura** /internal structure/ – części klasy i relacje między nimi (pokazuje ją diagram struktur złożonych)



Stereotyp /stereotype/

- Tworzy nową wersję klasy – z określonymi cechami i określonym użyciem.
- Przykłady:
  - **«*abstract*»** – klasa abstrakcyjna
    - lub nazwa klasy pisana kursywą,
    - lub napis {*abstract*} po nazwie klasy;
  - **«*interface*»** – interfejs klasy;
  - **«*enumeration*»** – enumeracja;
  - **«*boundary*»** – klasa z warstwy widoku;
  - **«*control*»** – klasa z warstwy kontrolera/prezentera;
  - **«*entity*»** – klasa z warstwy modelu;
  - **«*process*»** – aktywna klasa posiadająca swój proces;
  - **«*thread*»** – aktywna klasa posiadająca swój wątek;
- Klasa może mieć więcej niż 1 stereotyp.
- Można definiować własne stereotypy.

Atrybut /attribute, property/

- Własność cechująca klasę; przechowuje jej dane.
- **Składnia atrybutu:**

```
<property> ::= [‘«’<interface>’»’] [<visibility>] [‘/’]
<name> [‘:’ [<package>::]*<prop-type>] [‘[‘<multiplicity-range>‘]’]
[‘=’ <default>] [‘{‘<prop-modifier> [‘,’ <prop-modifier>]*’}’]

<visibility> ::= ‘+’ | ‘-‘ | ‘#’ | ‘~’

<prop-modifier> ::= ‘readOnly’ | ‘union’ | ‘subsets’ <property-name> |
‘redefines’ <property-name> | ‘ordered’ | ‘unordered’ | ‘unique’ |
‘nonunique’ | ‘seq’ | ‘sequence’ | ‘id’ | <prop-constraint>
```

<interface> – interfejs atrybutu;

<visibility> – widoczność atrybutu: publiczna (+), prywatna (-), chroniona (#) i pakietowa (~) (brak oznacza +);

/ oznacza atrybut pochodzi z innej klasy (pochodny);

<package> – nazwa pakietu;

<prop-type> – typ atrybutu;

<multiplicity-range> – liczność atrybutu (brak = 1, * = dowolnie wiele, od..do = zakres, ile);

<default> – domyślna wartość atrybutu;

<prop-modifier> – modyfikator atrybutu.

Atrybut

- Modyfikatory atrybutu:
  - **readOnly** – atrybut tylko do odczytu;
  - **union** – atrybut jest sumą jego podzbiorów;
  - **subsets <property-name>** – atrybut jest podzbiorem atrybutu o podanej nazwie;
  - **redefines <property-name>** – atrybut redefiniuje odziedziczony atrybut o podanej nazwie;
  - **ordered** – atrybut uporządkowany;
  - **unordered** – atrybut nieuporządkowany;
  - **unique** – atrybut niepowtarzalny;
  - **nonunique** – atrybut powtarzalny;
  - **seq** lub **sequence** – atrybut uporządkowany i powtarzalny;
  - **id** – atrybut jest częścią identyfikatora klasy;
  - **<prop-constraint>** – inne wyrażenie określające atrybut.
- Przykłady:

```
atrybut
atrybutAbstrakcyjny
+ liczba : int = 100
# punkty : figury::Punkt [*] {ordered}
```

Operacja /operation/

- Operacja, metoda, funkcja – to, co klasa wykonuje.
- **Składnia operacji:**

```
<operation> ::= [‘«’<interface>’»’] [<visibility>] <name>
‘(‘[<parameter-list>]‘)’ [‘:’ [<package>::]*[<return-type>]
[‘[‘<multiplicity-range>‘]’] [‘{‘<oper-property>[‘,’ <oper-property>]*‘}’]]

<visibility> ::= ‘+’ | ‘-‘ | ‘#’ | ‘~’

<parameter-list> ::= <parameter>[‘,’ <parameter>]*

<parameter> ::= [<direction>] <parameter-name> ’:’
[<package>::]*<type-expression> [’[’<multiplicity-range>’]’]
[’=’ <default>] [’{’<parm-property>[’,’ <parm-property>]*’}’]

<direction> ::= ’in’ | ’out’ | ’inout’

<parm-property> ::= ’ordered’ | ’unordered’ | ’unique’ | ’nonunique’ |
’seq’ | ’sequence’

<oper-property> ::= ‘redefines’ <oper-name> | ‘query’ | ‘ordered’ |
‘unordered’ | ‘unique’ | ‘nonunique’ | ‘seq’ | ‘sequence’ |
<oper-constraint>
```

Operacja

- Składnia operacji:

<interface> – interfejs operacji;

<visibility> – widoczność operacji: publiczna (+), prywatna (-), chroniona (#) i pakietowa (~) (brak oznacza +);

<package> – nazwa pakietu;

<return-type> – typ wyniku operacji;

<multiplicity-range> – liczność wyniku (brak = 1, * = dowolnie wiele, od..do = zakres, ile);

<paremeter> – parametr operacji;

<direction> – kierunek parametru operacji;

<type-expression> – typ parametru;

<default> – domyślna wartość parametru;

<direction> – kierunek parametru: wejściowy (*in*), wyjściowy (*out*), dowolny (*inout*) (brak oznacza *in*);

<parm-property> – modyfikator parametru operacji (jak modyfikator atrybutu);

<oper-property> – modyfikator operacji.

Operacja

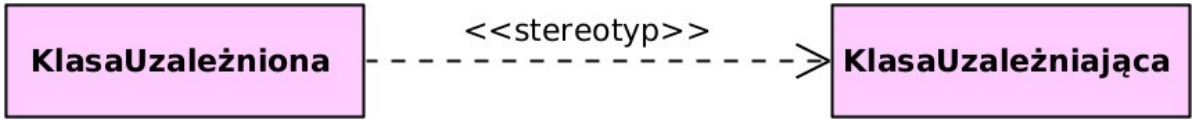
- Modyfikatory operacji:
  - **redefines <oper-name>** – operacja redefiniuje odziedziczoną operację o podanej nazwie;
  - **query** – operacja nie zmienia stanu systemu;
  - **ordered** – wynik operacji jest mnogi i uporządkowany;
  - **unordered** – wynik operacji jest mnogi i nieuporządkowany;
  - **unique** – wynik operacji jest mnogi i niepowtarzalny;
  - **nonunique** – wynik operacji jest mnogi i powtarzalny;
  - **seq** lub **sequence** – wynik operacji jest mnogi, uporządkowany i powtarzalny;
  - **<oper-constraint>** – inne wyrażenie określające operację.
- Przykłady:

```
operacja()
operacjaAbstrakcyjna()
- getLiczba(): int
+ setLiczba(ile: int): void
+ «create» Figura(punkt: figury::Punkt[4]): figury::Figura
# getPunkty(): figury::Punkt [*] {ordered}
```

Relacja zależności /dependency/

(przerywana linia z otwartym grotem)

- Najsłabsza relacja – wskazująca klasa „zna” wskazywaną klasę.
- Może mieć stereotyp, np.:



  - ***«call»*** – wskazująca klasa wywołuje metodę wskazanej klasy,
  - ***«create»*** – wskazująca klasa tworzy instancję wskazanej klasy,
  - ***«derive»*** – wskazująca klasa pochodzi od wskazanej klasy,
  - ***«instantiate»*** – wskazująca klasa zleca utworzenie instancji wskazanej klasy,
  - ***«bind»*** – wskazująca klasa implementuje wskazywany szablon,
  - ***«refine»*** – wskazująca klasa uszczegóławia wskazaną klasę,
  - ***«send»*** – wskazująca klasa wysyła sygnał do wskazanej klasy,
  - ***«trace»*** – zmiana wskazującej klasy jest uzależniona od wskazanej klasy (zwykle z innego modelu),
  - ***«use»*** – wskazująca klasa wymaga użycia wskazanej klasy (np. interfejsu).

Szablon klasy /template/

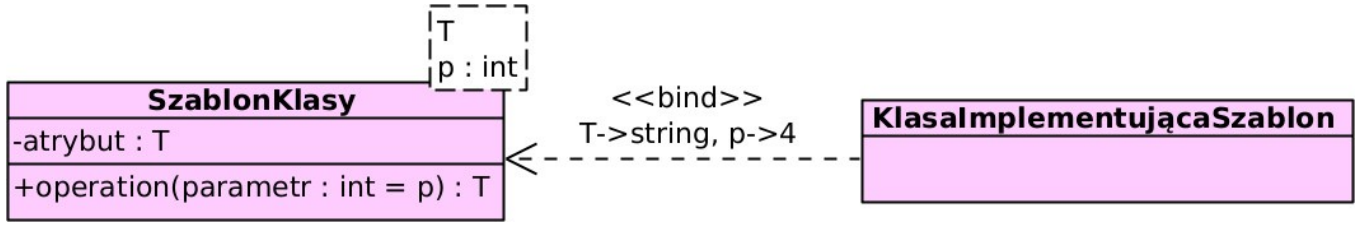
- **Klasa parametryzowana** /parameterized/:
  - definiuje i używa zbiór parametrów.
- **Parametr** – nieokreślony typ atrybutów i parametrów operacji:
  - konkretna implementacja tej klasy zamieni je na konkretne typy.
- **Składnia parametru:**

```
<template-parameter> ::= <template-param-name> [‘:’ <parameter-kind>]
[‘=’ <default>]
```

<parameter-kind> – typ parametru (m.in. metaklasa);

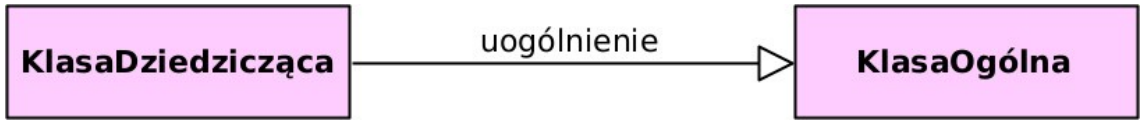
<default> – domyślna wartość parametru;

- Relacja zależności ze stereotypem **«*bind*»:**
  - wskazująca klasa implementuje wskazywany szablon klasy,
  - opisuje zamianę parametrów: `<template-param-name> -> <used-parameter>`.

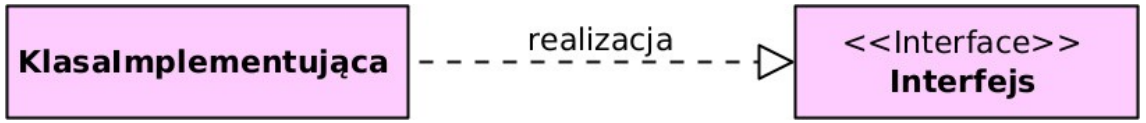


Relacje uogólnienia i realizacji

- Związek klasy podstawowej (ogólnej) z klasą pochodną (szczególną).
- **Relacja uogólnienia** /generalization/ (ciągła linia z grotem ∆).
  - Grot ∆ wskazuje klasę ogólną.
  - Wskazująca klasa dziedziczy po wskazywanej klasie.

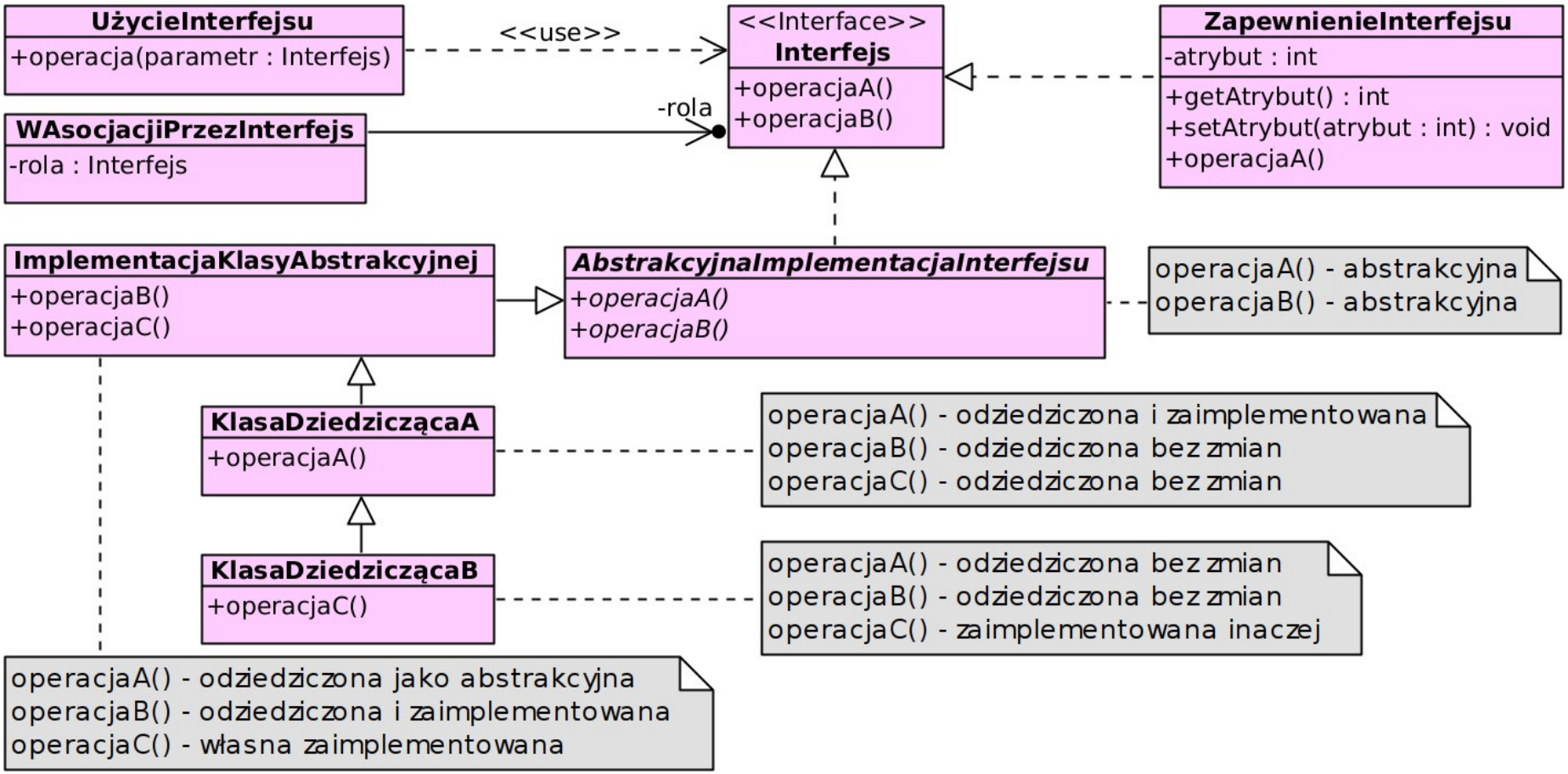


- **Relacja realizacji** /realization/ (przerywana linia z grotem ∆).
  - Grot ∆ wskazuje klasę ze stereotypem «interface».
  - Wskazująca klasa zapewnia (implementuje) wskazywany interfejs.

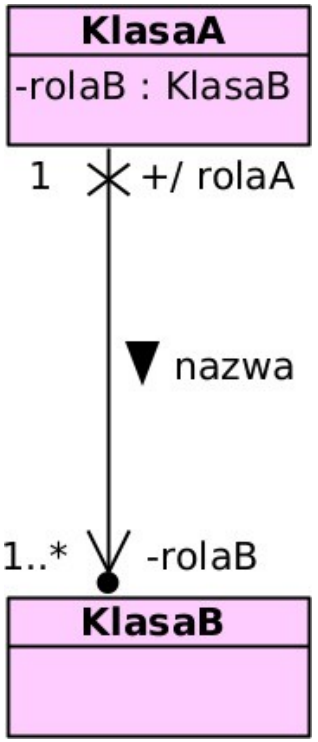


Przykład relacji uogólnienia i realizacji

- Założenie 1: klasa zapewniająca interfejs zawiera i pokazuje TYLKO implementowane operacje.
- Założenie 2: klasa dziedzicząca po klasie ogólnej zawiera wszystkie jej atrybuty i operacje, ale pokazuje TYLKO te, które zmienia.



Relacja asocjacji /association/

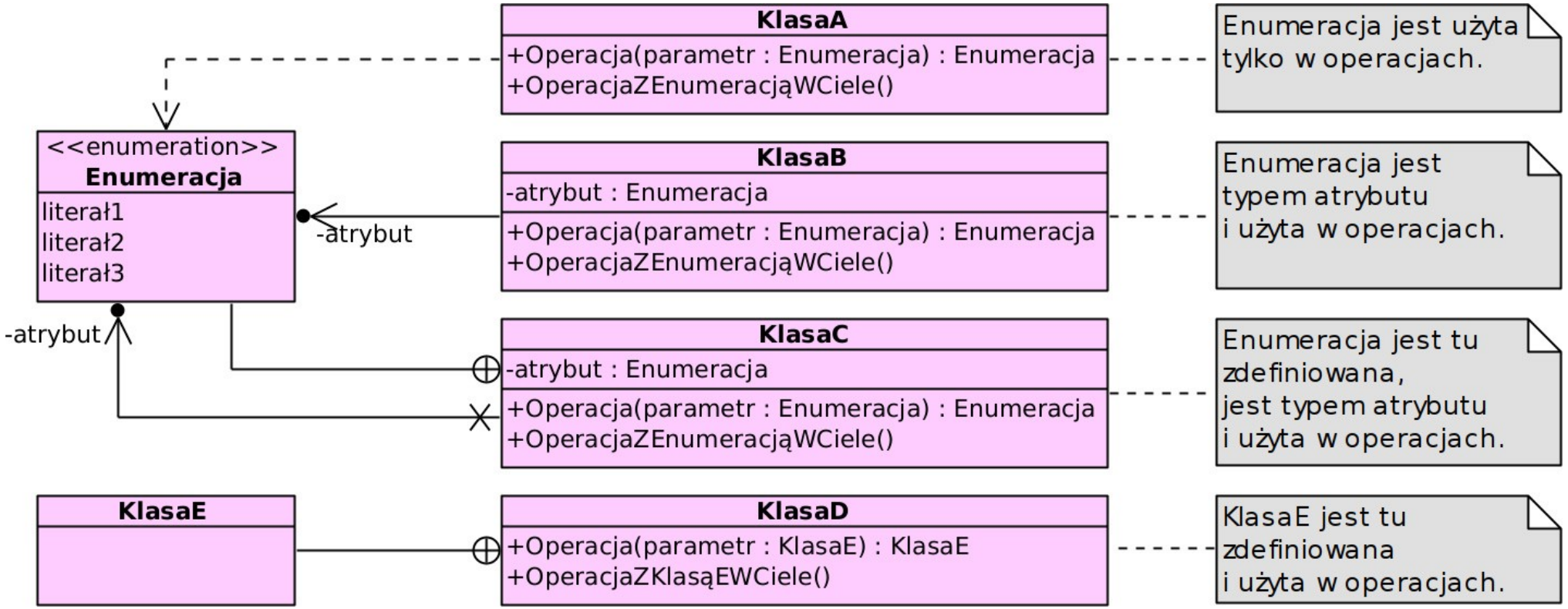


(ciągła linia z grotem otwartym lub bez)

- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez rolę – atrybut drugiej klasy lub związku).
- Otwarty grot wskazuje klasę, do której druga klasa ma dostęp:
  - brak grotu – dostęp nieokreślony,
  - brak grotu i znak **X** – brak dostępu do klasy z tym znakiem.
- Kropka • na końcu asocjacji – instancja klasy z tego końca jest atrybutem klasy z drugiego końca (rola = atrybut klasy):
  - brak – jest atrybutem asocjacji (rola ≠ atrybut klasy).
- Napis i znak ► przy środku asocjacji – jej nazwa i kierunek czytania.
- Napis przy końcu asocjacji (**rola**) – nazwa instancji klasy z tego końca dla klasy z drugiego końca.
  - może mieć znak widoczności (+ - # ~) i znak pochodności (/).
- Liczba lub przedział *min*..*max* przy końcu asocjacji – ile instancji klasy z tego końca jest związane z 1 instancją klasy z drugiego końca:
  - 1..* = co najmniej 1, 0..* (*) = dowolnie wiele; brak ≈ 1.

Przykład relacji zależności, asocjacji i posiadania (i przykład enumeracji)

- Relacje **KlasaB** → **Enumeracja** oraz **KlasaC** → **Enumeracja** są strukturalne (dostęp przez **atrybut** klasy).
- **Enumeracja** jest zdefiniowana w **KlasieC**.
- **KlasaE** jest zdefiniowana w **KlasieD**.



Relacja agregacji /aggregation/

(asocjacja z grotem ◊)

- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez rolę – atrybut drugiej klasy lub związku).
- Relacja asocjacji niezależna całość–niezależna część:
  - instancja klasy „całość” składa się z instancji klasy „część”,
  - „część” może istnieć samodzielnie,
  - „część” NIE jest wyłączną własnością „całości”.
- Grot ◊ wskazuje klasę „całość”.
- Otwarty grot wskazuje klasę „część”.

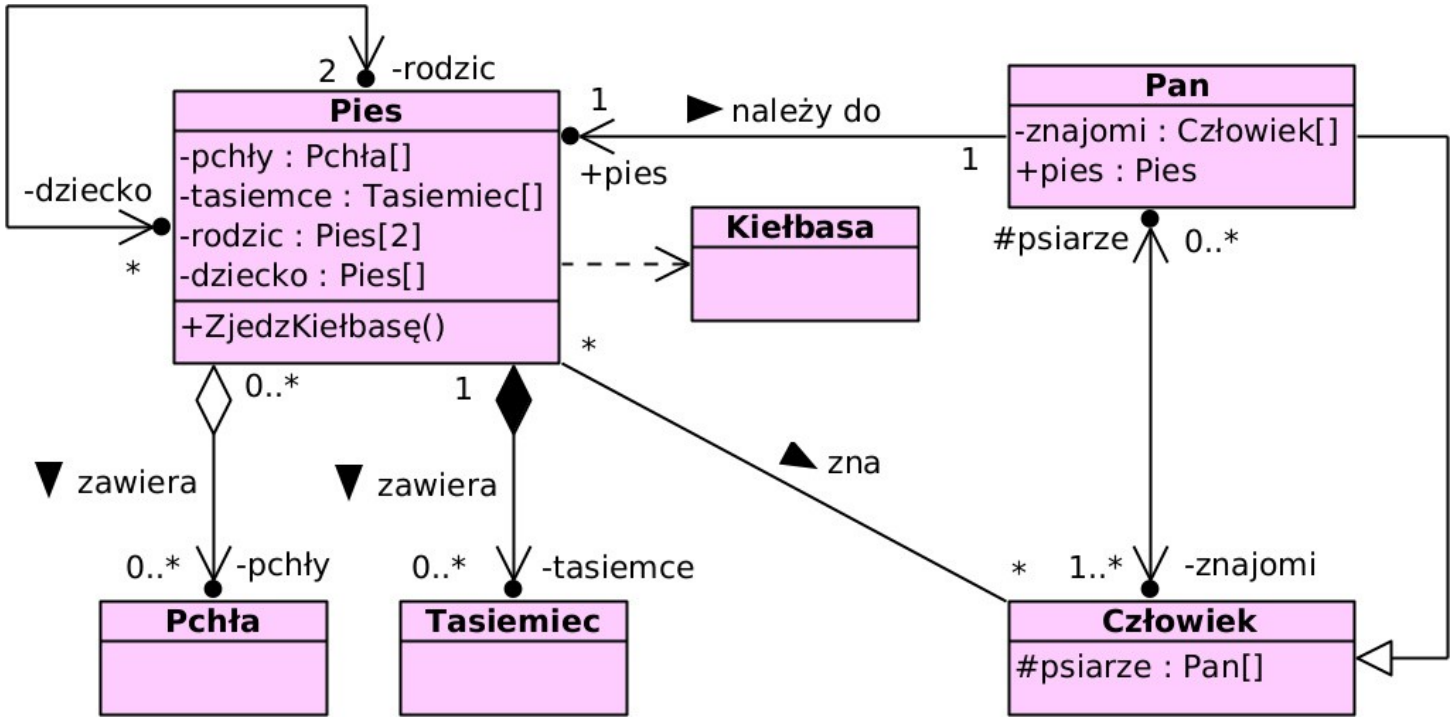
Relacja kompozycji /aggregation/

(asocjacja z grotem ♦)

- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez rolę – atrybut drugiej klasy lub związku).
- Relacja asocjacji uzależniona całość–zależna część:
  - instancja klasy „całość” składa się z instancji klasy „część”,
  - „część” NIE może istnieć samodzielnie,
  - „część” jest wyłączną własnością „całości”.
- Grot ♦ wskazuje klasę „całość”.
- Otwarty grot wskazuje klasę „część”.

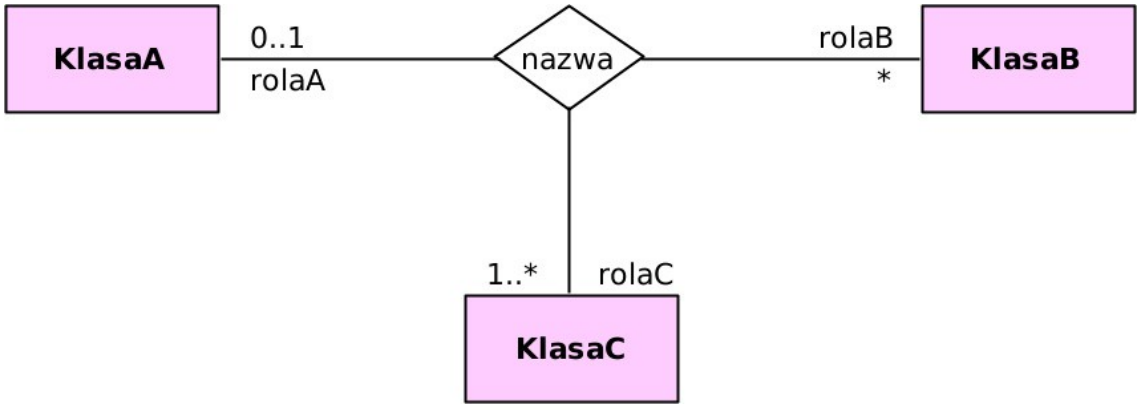
Przykład relacji asocjacji, agregacji i kompozycji

- **Pies** może mieć **pchły** i **tasiemce** (wspólnie tworzą ekosystem).
- **Pchła** może też żyć samotnie lub też w relacji z kilkoma **Psami**.
  - Gdy **Pies** zdycha, jego **pchły** mogą żyć dalej (np. na innym psie).
- **Tasiemiec** nie może żyć samotnie i musi być w relacji z 1 **Psem**.
  - Gdy **Pies** zdycha, jego **tasiemce** zdychają z nim lub… „coś” przenosi je do innego psa.
- **Pies** nie ma dostępu do **Człowieka** ani do **Pana**.
- **Pan** ma **Psa** (ale **Pies** nie jest jego częścią).



Relacja asocjacji n-krotnej /n-ary association/

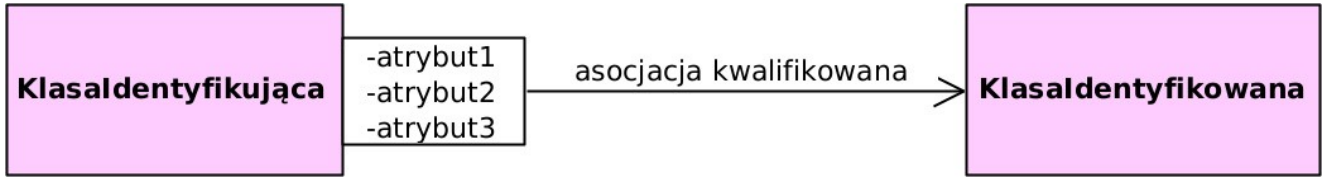
- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez rolę – atrybut innej klasy lub związku).
- Relacja wzajemnej asocjacji n klas (n ≥ 3).
- Węzeł <> definiuje nazwę n-krotnej asocjacji.
- Każda klasa łączy się zwykłą asocjacją z tym węzłem (w stosunku x do 1 po stronie węzła).



Relacja asocjacji kwalifikowanej /qualified association/

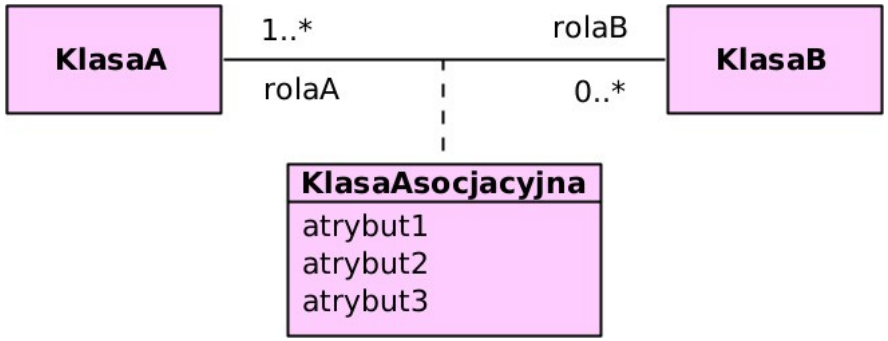
- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez kwalifikator).
- **Kwalifikator** – zbiór atrybutów końca asocjacji:
  - umieszczony w ramce na końcu asocjacji,
  - jednoznacznie identyfikuje instancję klasy z drugiego końca.
- Składnia atrybutu:

```
<property> ::= [<visibility>] ['/'] <name> [':' [<package>::]*<prop-type>]
```

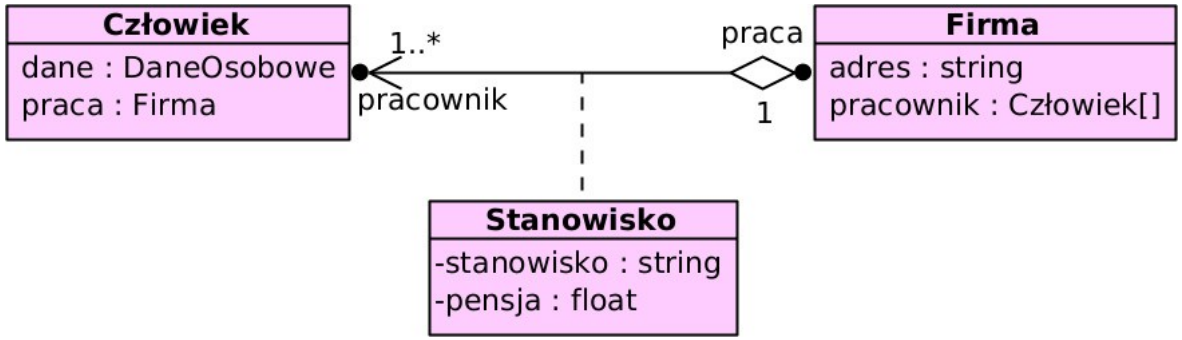


Klasa asocjacyjna /association class/

- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez klasę asocjacyjną).
- Łączy się z asocjacją relacją ograniczenia (przerywana linia):
  - w miejscu połączenia może być węzeł <> z nazwą klasy asocjacyjnej.
- Definiuje atrybuty asocjacji:
  - Jej instancja jednoznacznie identyfikuje związek konkretnych instancji klas powiązanych asocjacją.

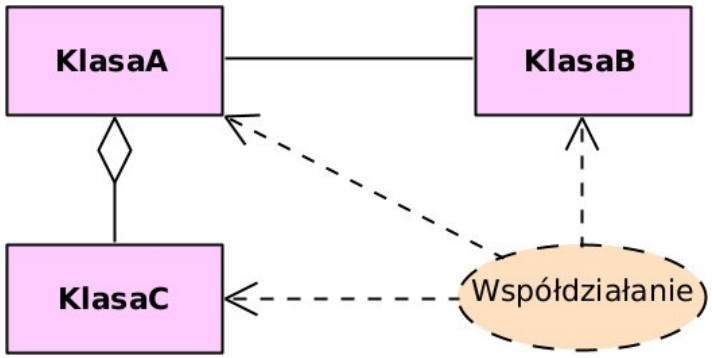


- **Przykład**



Współdziałanie /collaboration/

- Współdziałanie wybranych klas
  - w celu osiągnięcia wspólnych celów (zadań).
- Klasyfikator ze stereotypem «collaboration» i nazwą współdziałania
  - lub owal z przerywaną krawędzią i nazwą współdziałania.
- Łączy się z klasą relacją zależności.
  - Wskazuje na klasy potrzebne do osiągnięcia celu.
- Role klas, wiążące je relacje i szkic ich użycia pokazuje **diagram struktur złożonych.**



3

Obiektowe modelowanie struktury programu

Podstawowe założenia

- **Klasy i obiekty** (instancje klas):
  - reprezentują elementy rzeczywistego świata;
  - są definiowane na podstawie tego, za co są odpowiedzialne;
  - wykonują operacje, komunikują się, mają atrybuty i zmieniają swój stan.
- **Abstrakcja** – ogólna definicja klasy i operacji:
  - określa tylko funkcjonalność,
  - pomija szczegóły implementacji,
  - wydziela najważniejsze cechy obiektu lub rodziny obiektów,
  - przy pomocy abstrakcyjnych klas i operacji oraz interfejsów.
- **Dziedziczenie** – definiowanie klasy na bazie innej, ogólniejszej klasy:
  - kopiuje wybrane atrybuty i operacje klasy bazowej,
  - redefiniuje wybrane atrybuty i operacje klasy bazowej,
  - dodaje własne atrybuty i operacje.

Podstawowe założenia

- **Polimorfizm** – wielopostaciowość klasy bazowej (klasy dziedziczonej lub zapewnianego interfejsu):
  - różne zachowanie obiektów różnych klas klasy bazowej (gdy są używane jako obiekty klasy bazowej),
  - przez zapewnienie tego samego interfejsu lub dziedziczenie tej samej klasy.
- **Hermetyzacja** – ograniczenie dostępu do klasy i obiektu:
  - ukrywa szczegóły implementacji klasy;
  - określa dostępność atrybutów i operacji klasy z zewnątrz;
  - ogranicza możliwość zmiany stanu obiektu przez inny obiekt (tylko on sam ma pełny dostęp do swoich atrybutów i operacji);
  - przy pomocy widoczności atrybutów i operacji.

Identyfikacja klas, obiektów, atrybutów i operacji

- **Analiza słownej specyfikacji** wymagań i przypadków użycia:
  - rzeczownik może określać klasę, jej instancję lub atrybut;
  - czasownik może określać operację klasy.
- **Nazwa** klasy, atrybutu i operacji – jednoznacznie zrozumiała i określająca przeznaczenie.
- **Dostępność** do klasy, atrybutu i operacji (widoczność).
- **Spójność klasy** – wykonuje tylko swoje zadania.
- **Rozdzielenie operacji** na tworzące obiekt i używające obiekt.
- **Normalizacja operacji** klasy – różna funkcjonalność różnych operacji klasy.
- **Stereotyp klasy** – zastosowanie odpowiedniego typu klasy.
- **Kompozycja i dekompozycja klas**:
  - podzielenie klasy, która ma za dużo zobowiązań, na mniejsze klasy;
  - połączenie klas, które mają za mało zobowiązań, w większą klasę.

Identyfikacja klas, obiektów, atrybutów i operacji

na podst. UML Przewodnik użytkownika, G. Booch, J. Rumbaugh, I. Jacobson; wyd. WNT i Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr, wykład 3

- Zdefiniuj klasy, które współpracują ze sobą w celu wykonania danej czynności.
  - Zdefiniuj sposoby współpracy między klasami.
- Zdefiniuj i rozdziel zobowiązania współpracujących ze sobą klas.
  - Klasy, które mają za dużo zobowiązań, podziel na mniejsze.
  - Klasy, które mają za mało zobowiązań połącz w większe.

Identyfikacja relacji między klasami i obiektami

- **Zależność** (związek niestrukturalny):
  - gdy operacja klasy ma dostęp do innej klasy, ale NIE przez swój atrybut (związek niestrukturalny).
- **Związek strukturalny**:
  - **asocjacja** – gdy operacja klasy ma dostęp do innej klasy przez swój atrybut,
  - **agregacja** / **kompozycja** – gdy dodatkowo klasa zawiera instancję tej drugiej klasy (nie na wyłączność / na wyłączność),
  - określa ilościowy stosunek między instancjami powiązanych klas,
  - określa jaką rolę pełnią instancje powiązanych klas względem siebie.
- **Uogólnienie**:
  - gdy różne klasy mają wspólne atrybuty, operacje lub relacje;
  - ich wspólna część trafia do ich klasy bazowej (dziedziczonej);
  - łańcuch dziedziczenia NIE może się zapętlać.

Identyfikacja relacji między klasami i obiektami

na podst. UML Przewodnik użytkownika, G. Booch, J. Rumbaugh, I. Jacobson; wyd. WNT i Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr, wykład 3

- Gdy między klasami zachodzi związek strukturalny:
  - połącz je relacją asocjacji, agregacji lub kompozycji;
  - zdefiniuj stosunek liczbowy między klasami i ich role.
  - Jeśli jedna klasa jest strukturalną lub organizacyjną częścią drugiej klasy, użyj agregacji lub kompozycji;
  - w przeciwnym wypadku użyj asocjacji.
- Gdy między klasami zachodzi tylko związek niestrukturalny:
  - połącz je relacją zależności;
  - jeśli można, określ stereotyp zależności.

Identyfikacja relacji między klasami i obiektami

na podst. UML Przewodnik użytkownika, G. Booch, J. Rumbaugh, I. Jacobson; wyd. WNT
i Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr, wykład 3

- Znajdź zobowiązania, atrybuty i operacje wspólne dla różnych klas.
  - Utwórz dla nich klasę ogólniejszą i przenieś je do niej.
- Klasę ogólniejszą połącz z klasami po niej dziedziczącymi relacją uogólnienia:
  - tylko wtedy, gdy klasa dziedzicząca jest „rodzajem” klasy ogólniejszej.
  - Unikaj cyklicznych uogólnień (dziecko nie może być swoim przodkiem).
  - Ogranicz liczbę poziomów uogólnień do kilku.

Identyfikacja wzorców projektowych

- **Wzorzec projektowy** – koncepcyjny model rozwiązania powtarzającego się problemu projektowego:
  - **złożony** – wiąże ze sobą warstwy oprogramowania,
  - **prosty** – wiąże ze sobą klasy (zwykle w ramach jednej warstwy oprogramowania).

SOLID – założenia programowania obiektowego

- **Jedna odpowiedzialność** (**S**ingle responsibility):
  - klasa / operacja powinna odpowiadać tylko za 1 zadanie.
- **Otwartość / zamkniętość** (**O**pen / Closed):
  - klasa / operacja powinna być
    otwarta na rozszerzenia i zamknięta na modyfikacje,
    więc: jej zachowanie nie powinno wymagać zmiany jej kodu.
- **Podstawienie Liskov'ej** (**L**iskov substitution):
  - operacja używająca klasę bazową
    powinna móc używać klasy po niej dziedziczące, bez ich znajomości.
- **Segregacja interfejsów** (**I**nterface segregation):
  - interfejsy powinny być dedykowane zadaniom, a nie ogólne.
- **Odwrócenie zależności** (**D**ependency inversion):
  - klasa / operacja powinna zależeć od abstrakcji klasy / operacji,
    a nie od konkretnych implementacji klas / operacji.

Szczegółowość modelu

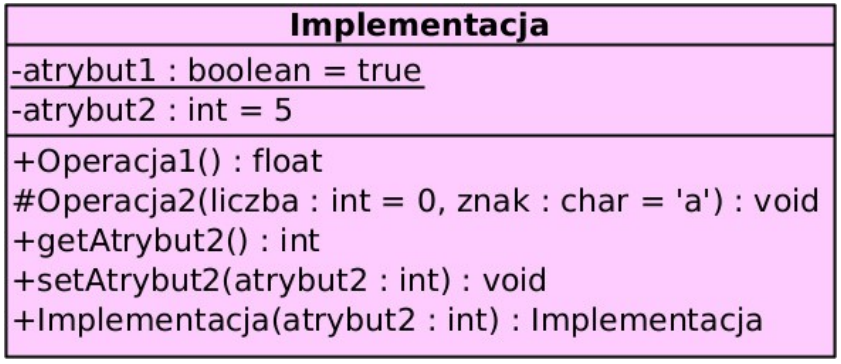
- **Zbyt ogólny model**
  - zbyt dowolna i nieudokumentowana implementacja modelu,
  - brak wykrycia bloków ponownego użycia.
- **Zbyt szczegółowy model**
  - zbyt sztywna i skomplikowana implementacja modelu,
  - nieistotne szczegóły utrudniają analizę modelu.

Poziomy szczegółowości klasy

- **Koncepcja** – definicja zadań klasy:
  - ogólna koncepcja klasy w modelu domeny,
  - nazwanie klasy, głównych atrybutów i głównych operacji.

- **Specyfikacja** – definicja zachowania klas:
  - wstępna, ogólna definicja atrybutów, operacji, stereotypów i relacji klasy.

- **Implementacja** – definicja „kodu” klasy:
  - końcowa, pełna definicja atrybutów, operacji, stereotypów i relacji klasy.



4

Przykłady

Przykłady

zobacz: Inżynieria Oprogramowania, Z. Kruczkiewicz, PWr, wykład 3

1. Identyfikacja klas. (strona 60—67).

Temat następnej prezentacji

Modelowanie struktury – diagram pakietów, diagram obiektów i diagram struktur złożonych