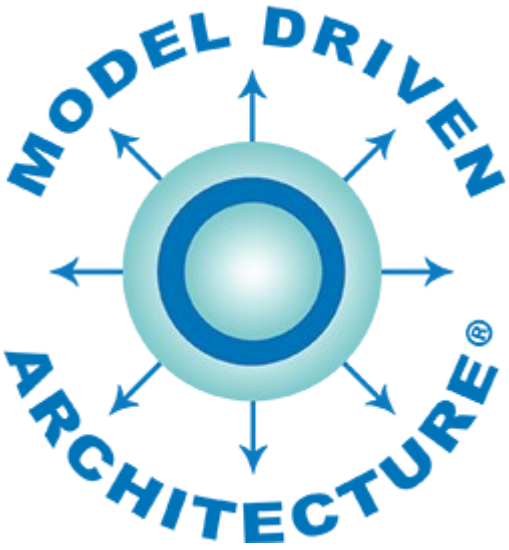


Architektura Sterowana Modelem

Model Driven Architecture

prezentacja 4

Język UML – diagramy struktury

wersja 1.0

dr inż. Paweł Głuchowski

Wydział Informatyki i Telekomunikacji, Politechnika Wrocławska

Treść prezentacji

1. Klasyfikatory
2. Diagram pakietów
3. Diagram klas
4. Diagram obiektów
5. Diagram struktur złożonych
6. Diagram komponentów
7. Diagram wdrożenia
8. Diagram profilu

1

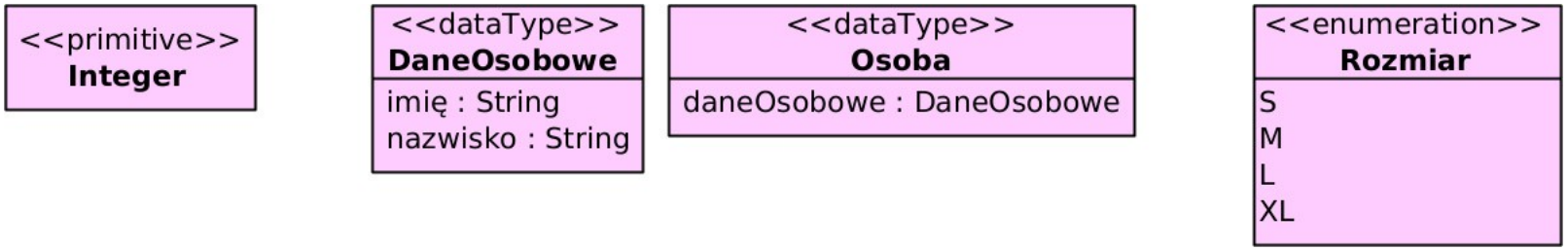
Klasyfikatory

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/

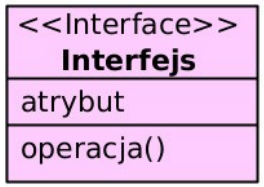
- Modeluje asynchroniczną komunikację między obiektami (powoduje reakcję obiektu, ale nie powoduje odpowiedzi).
- Ma stereotyp: «*signal*».
- Jego atrybuty to treść komunikacji.
- Może być parametryzowany /parameterized/ i ograniczony /bound/.
- Może być parametrem szablonu /template parameter/.

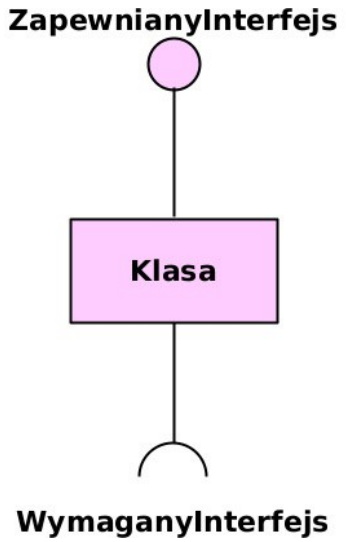
Odbiór /reception/

- 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/

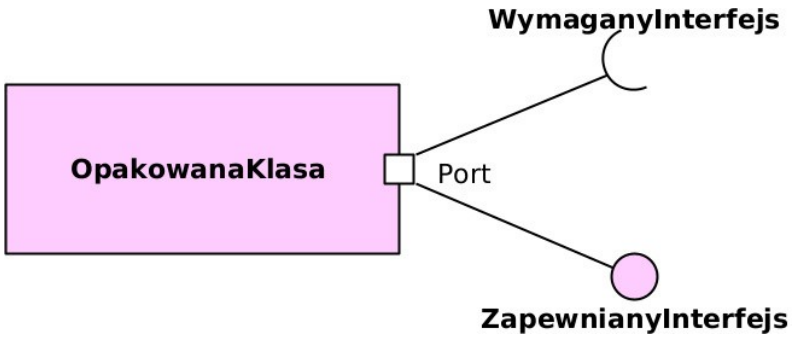
- Abstrakcja klasyfikatora (np. klasy) – jego publiczne atrybuty, operacje i wewnętrzna struktura.
  - Ogranicza zapewniającą go (implementującą) klasę (określa, co klasa ma zaimplementować).
  - NIE określa sposobu implementacji.
  - NIE ma instancji.
- Ma stereotyp: «*interface*»
  lub jest to „lizak” (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 pakietów

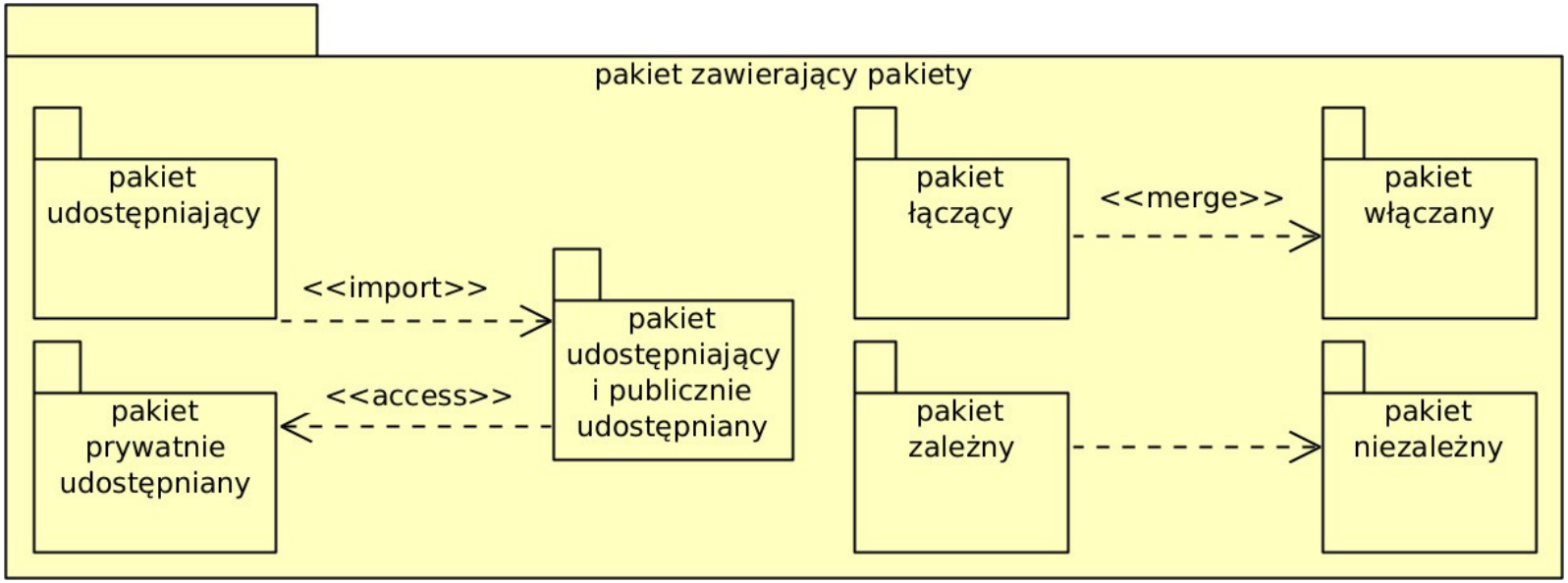
Diagram pakietów /package diagram/

- Modeluje logiczne uporządkowanie innych elementów:
  - gdzie ten element się znajduje,
  - w jakiej relacji jest ten element z innym elementem.

- Inne diagramy też mogą zawierać pakiety i związki między nimi, np.:
  - diagram wymagań,
  - diagram przypadków użycia,
  - diagram klas,
  - diagram komponentów.

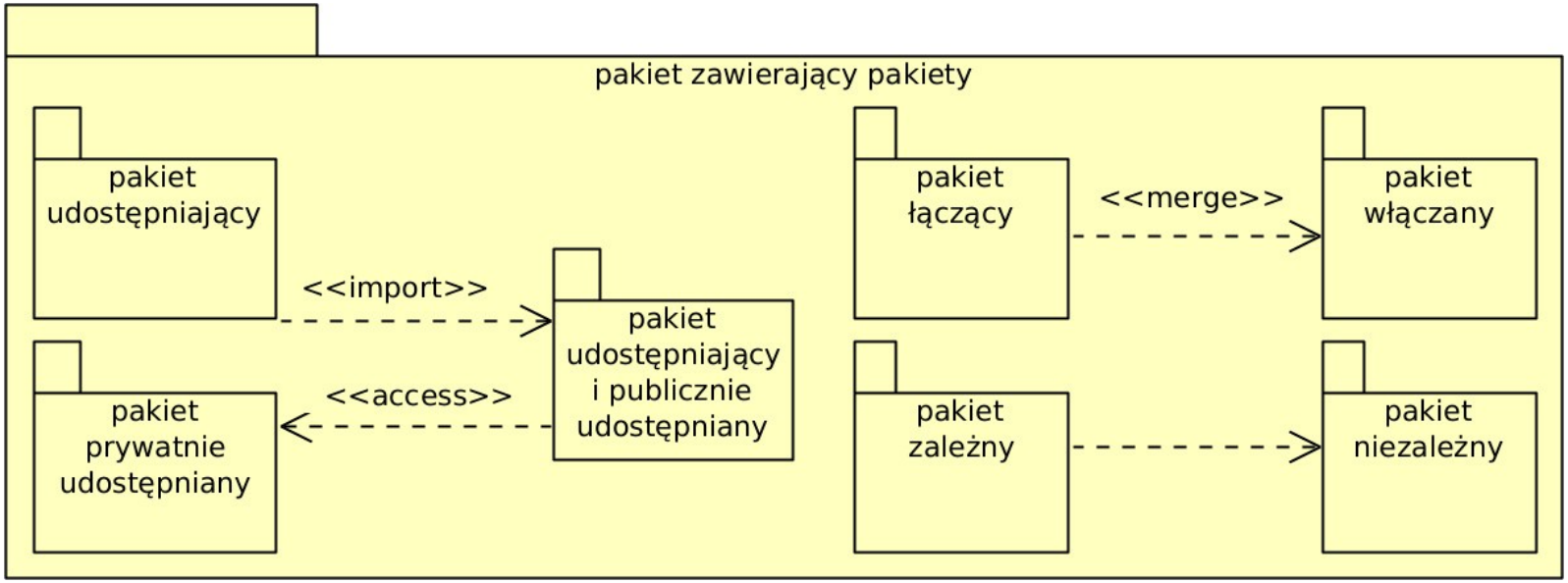
Relacje między pakietami

- Pakiety mogą być **zagnieżdżone.**
- **Relacja dostępu** – elementy jednego pakietu używają publicznych elementów innego pakietu:
  - **relacja *«access»*** – nazwy elementów wskazanego pakietu dodawane są do przestrzeni nazw używających elementów wskazującego pakietu jako prywatne;
  - **relacja *«import»*** – nazwy elementów wskazanego pakietu dodawane są do przestrzeni nazw używających elementów wskazującego pakietu jako publiczne.



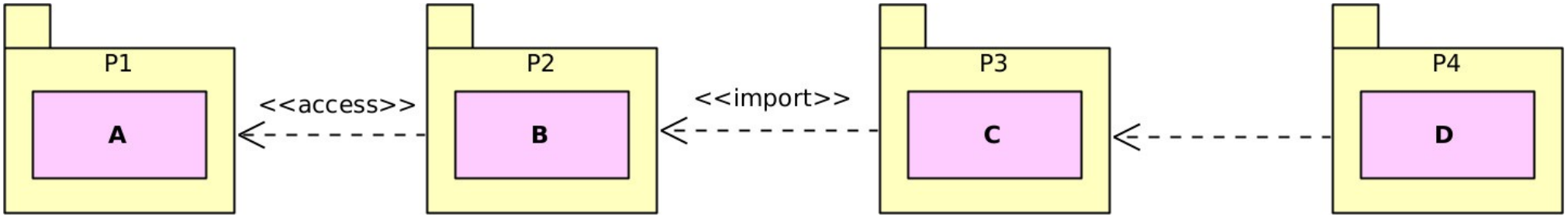
Relacje między pakietami

- **Relacja łączenia** – elementy jednego pakietu są rozbudowywane o nazwą i funkcjonalnością odpowiadające im elementy innego pakietu:
  - **relacja *«merge»*** – zawartość elementów wskazanego pakietu łączona jest z zwartością odpowiadających im elementów wskazującego pakietu.
- **Relacja zależności** – w innym przypadku (słabsza lub nieokreślona relacja):
  - relacja bez stereotypu «» – elementy wskazującego pakietu wymagają specyfikacji lub użycia elementów wskazanego pakietu.



Przykład relacji dostępu

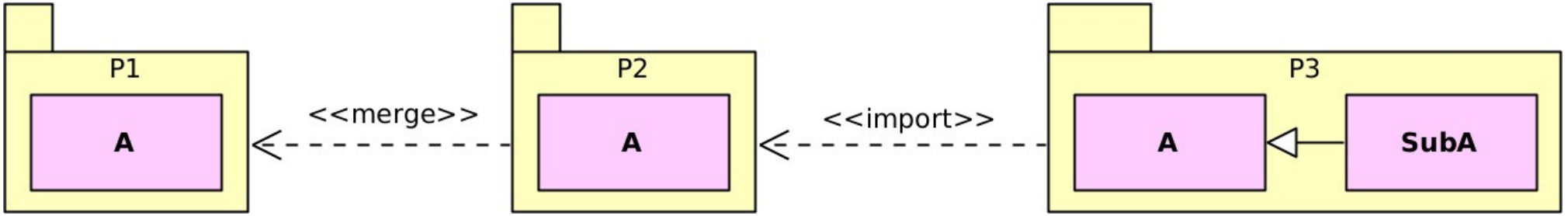
- Założenie: klasy **A**, **B** i **C** są publiczną zawartością swoich pakietów.
- Klasa **B** z pakietu **P2**:
  - ma dostęp do klasy **A** z pakietu **P1** (**P1::A**) jako prywatną klasę **P2::A** (P2 → *«access»* → P1).
- Klasa **C** z pakietu **P3**:
  - nie ma dostępu do klasy **P2::A** (jest prywatna),
  - ma dostęp do klasy **B** z pakietu **P2** (**P2::B**) jako publiczną klasę **P3::B** (P3 → *«import»* → P2).
- Klasa **D** z pakietu **P4:**
  - ma dostęp do klas **P3::B** i **P3::C** (są publiczne).



Przykład relacji łączenia i dostępu

na podst. Unified Modeling Language (UML)

- Pakiet **P2** definiuje nadbudowę klasy **A**, która jest zdefiniowana w pakiecie **P1** (P2 → *«merge»* → P1).
  - **P2** włącza do siebie zawartość **P1** (włącza **P1::A** do **P2::A**).
- Pakiet **P3** definiuje klasę **SubA** – podklasę klasy **A** z pakietu **P2** (uogólnienie i P3 → *«import»* → P1).
  - **A** w pakiecie **P3** (**P3::A**) to **A** z pakietu **P2** (**P2::A**).
  - **A** w pakiecie **P2** (**P2::A**) to wynik włączenia **P1::A** do **P2::A**, a nie sama nadbudowa klasy **A** (**P2::A**).



3

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).

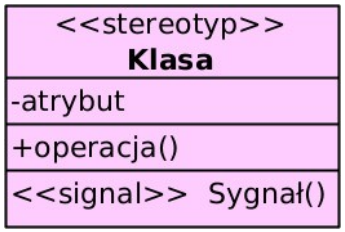
Składa się z: atrybutów, operacji, odbiorów, portów i relacji.

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

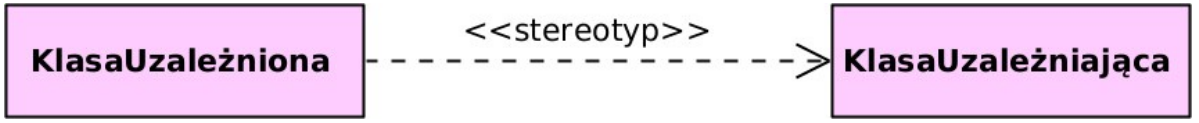
- Modyfikatory operacji:
  - **redefines <oper-name>** – operacja redefiniuje odziedziczoną operację o podanej nazwie;
  - **query** – operacja nie zmienia stanu systemu;
  - **ordered** – wynik operacji jest mnogi i uporządkowany;
  - **unordered** – wynik operacji jest mnogi i nieuporządkowany;
  - **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/

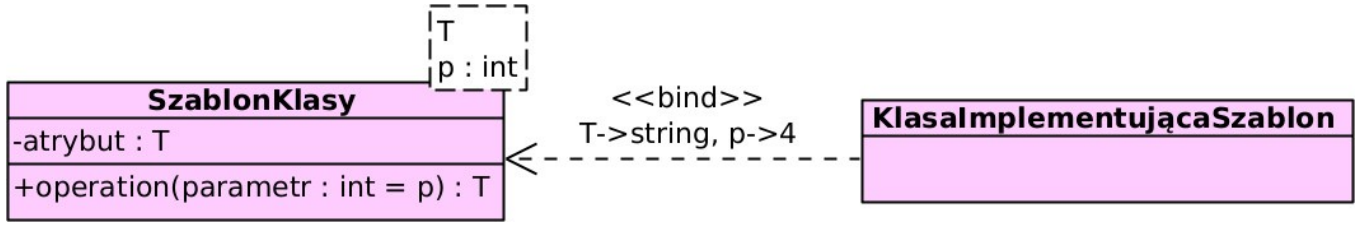
- **Klasa parametryzowana** /parameterized/:
  - definiuje i używa zbiór parametrów.
- **Parametr** – nieokreślony typ atrybutów i parametrów operacji:
  - konkretna implementacja tej klasy zamieni je na konkretne typy.
- **Składnia parametru:**

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

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

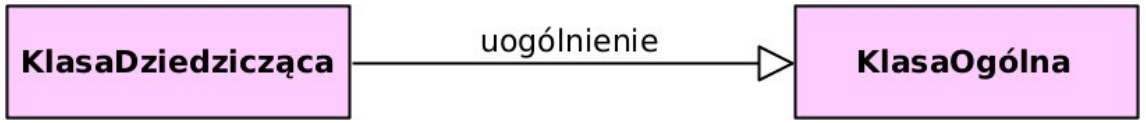
<default> – domyślna wartość parametru;

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

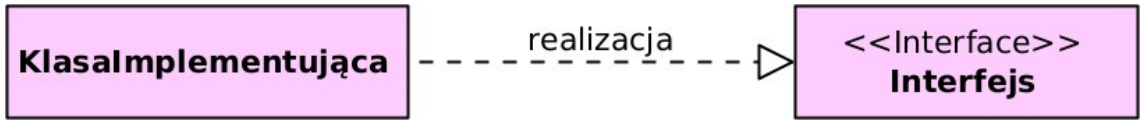


Relacje uogólnienia i realizacji

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

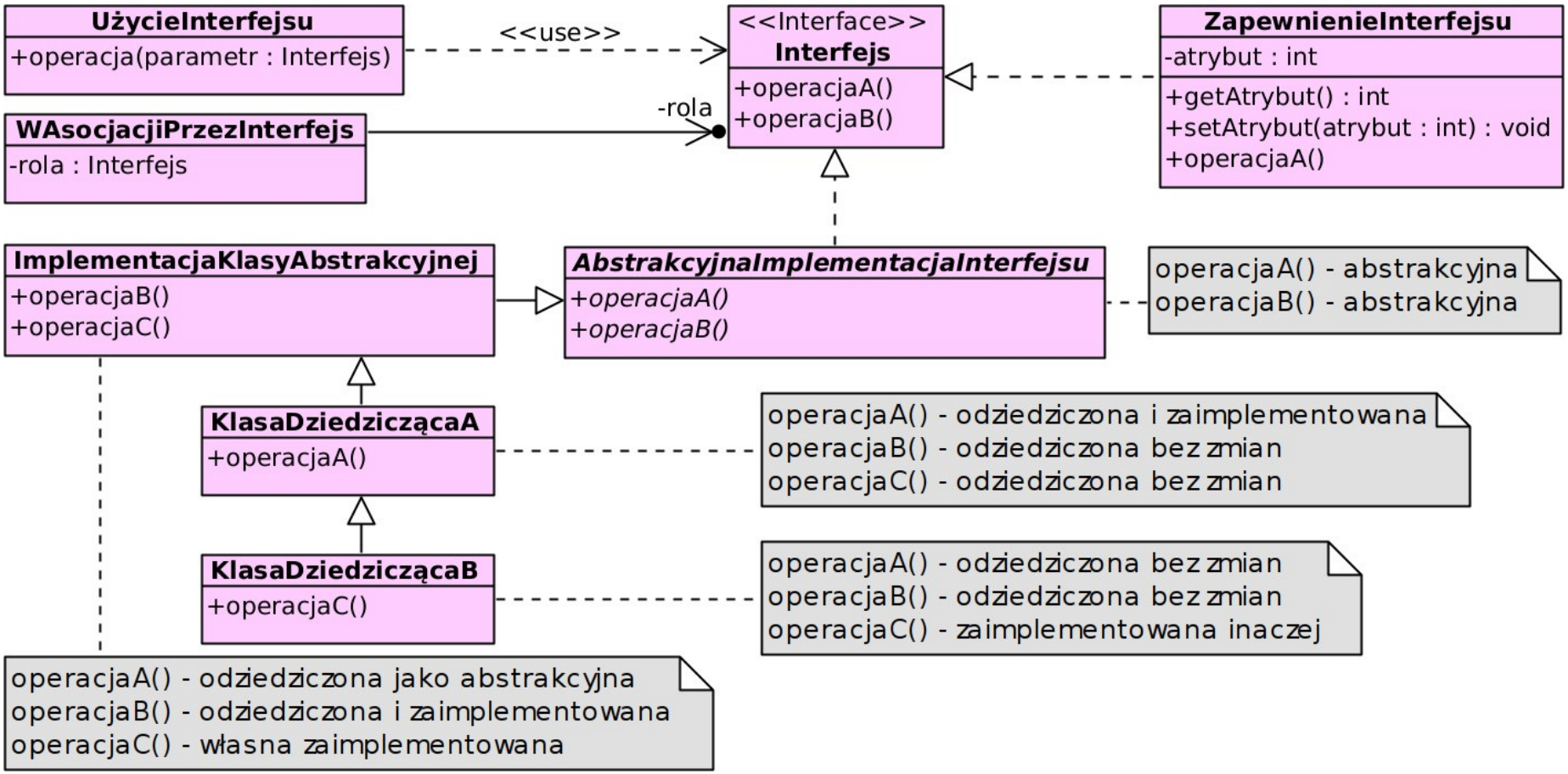


- **Relacja realizacji** /realization/ (przerywana linia z grotem Δ).
  - Grot Δ wskazuje klasę ze stereotypem «interface».
  - Wskazująca klasa zapewnia (implementuje) wskazywany interfejs.

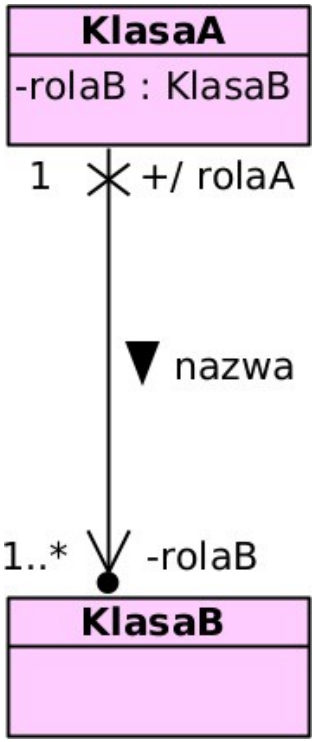


Przykład relacji uogólnienia i realizacji

- 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/

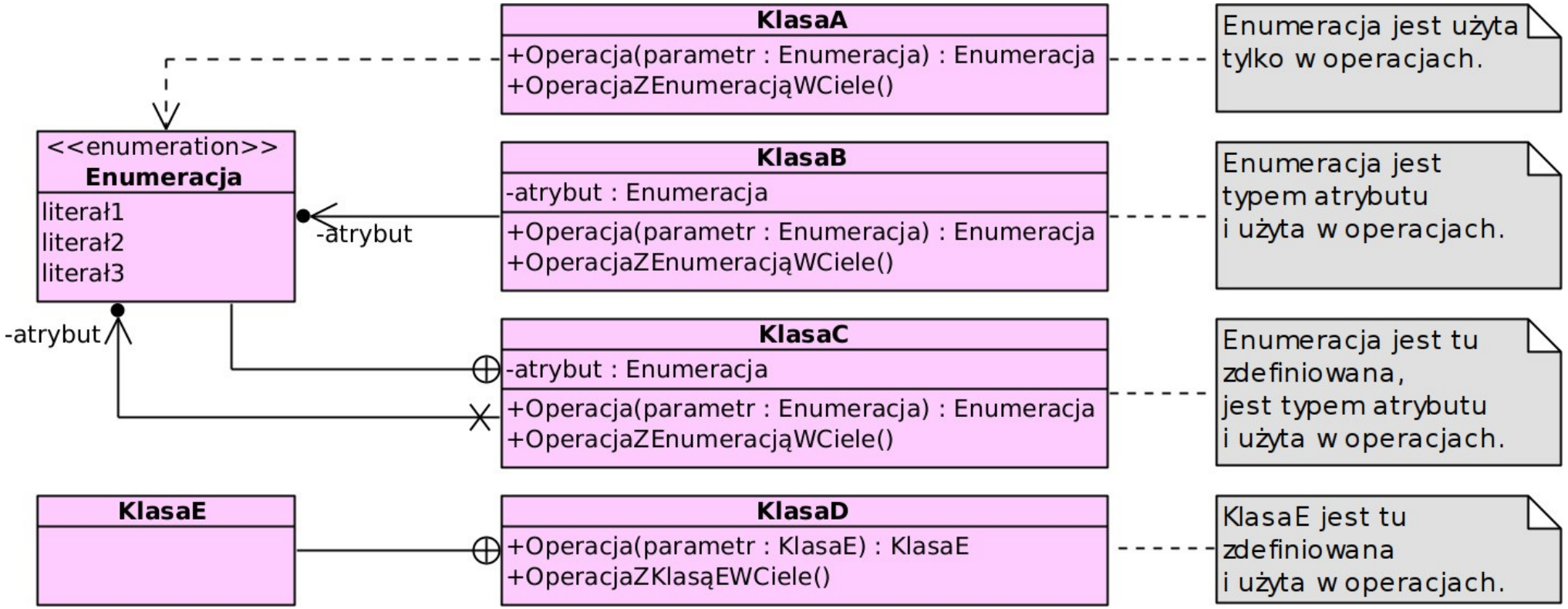


(ciągła linia z grotem otwartym lub bez)

- 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”.

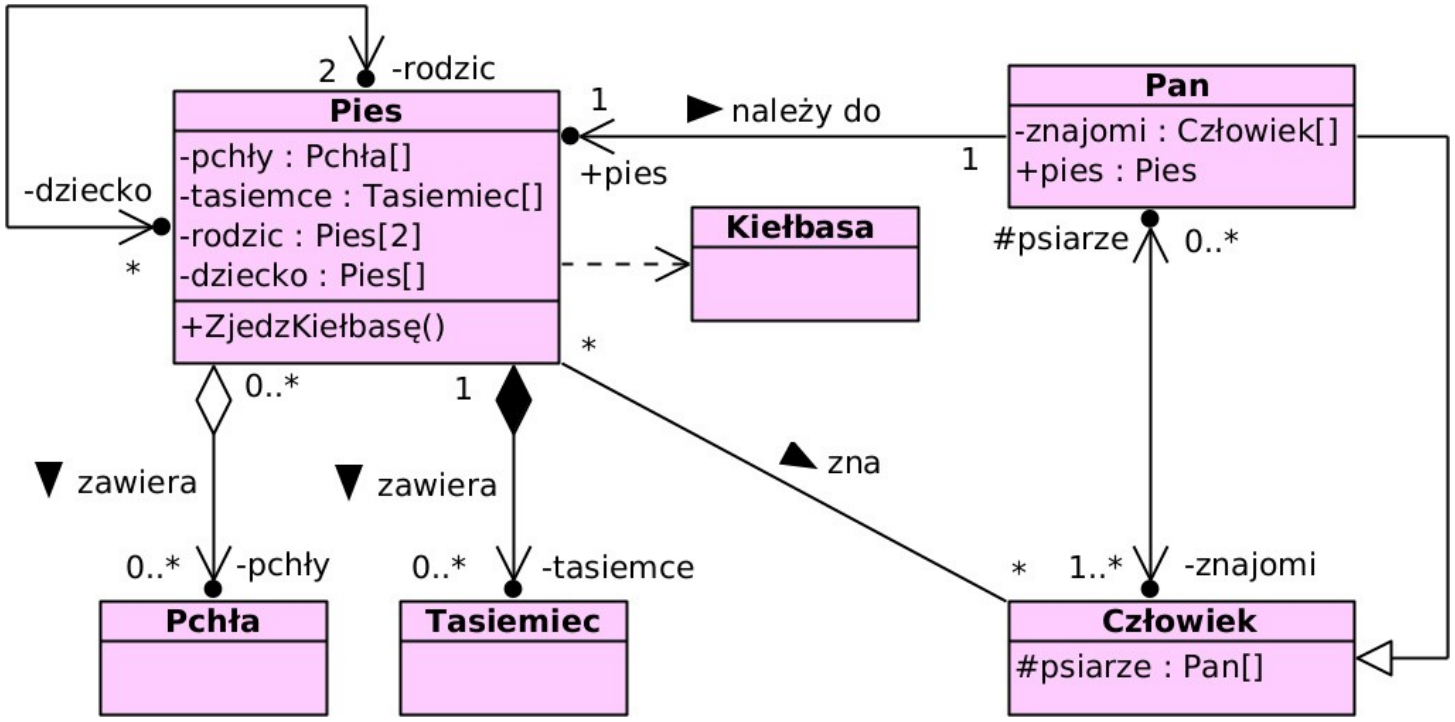
Relacja kompozycji /aggregation/

(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”.

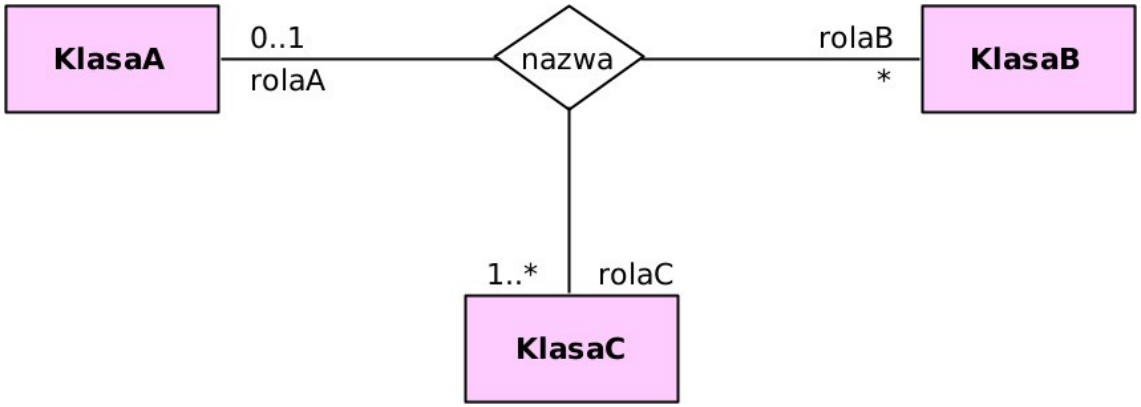
Przykład relacji asocjacji, agregacji i kompozycji

- **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/

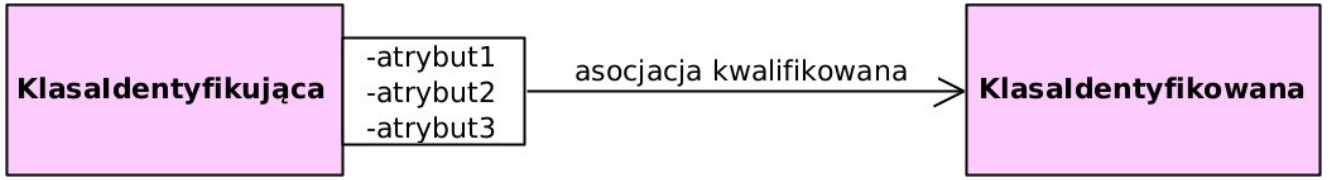
- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez rolę – atrybut innej klasy lub związku).
- Relacja wzajemnej asocjacji n klas (n ≥ 3).
- Węzeł <> definiuje nazwę n-krotnej asocjacji.
- Każda klasa łączy się zwykłą asocjacją z tym węzłem (w stosunku x do 1 po stronie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



Relacja asocjacji kwalifikowanej /qualified association/

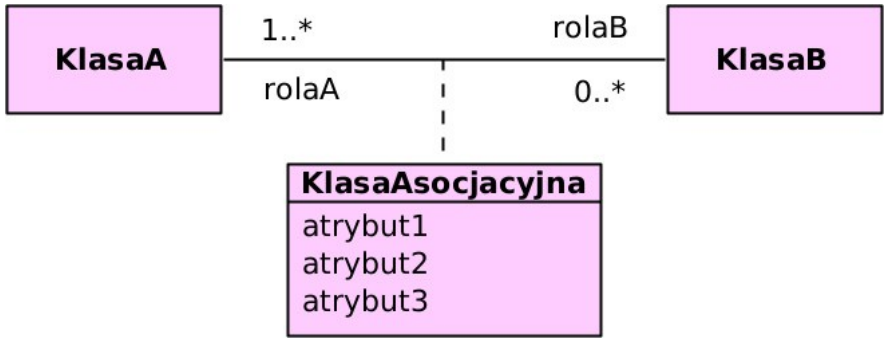
- Strukturalny związek klas i stosunek ilościowy ich instancji (dostęp do klasy przez kwalifikator).
- **Kwalifikator** – zbiór atrybutów końca asocjacji:
  - umieszczony w ramce na końcu asocjacji,
  - jednoznacznie identyfikuje instancję klasy z drugiego końca.
- Składnia atrybutu:

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

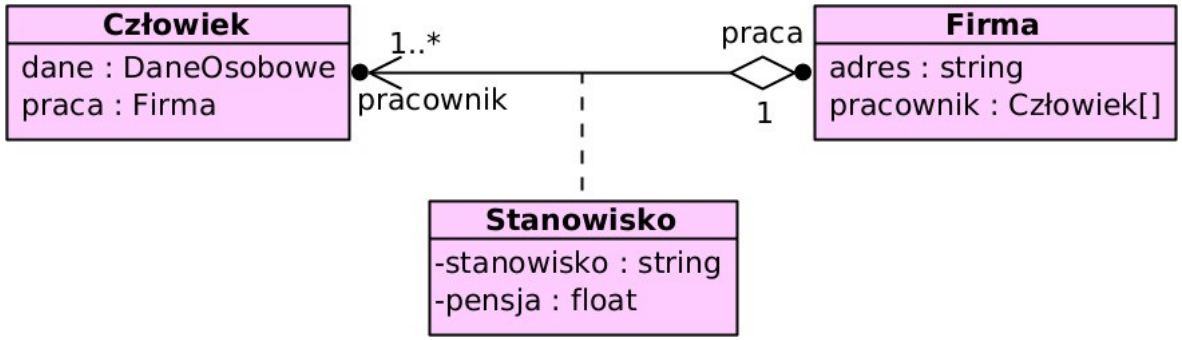


Klasa asocjacyjna /association class/

- 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.

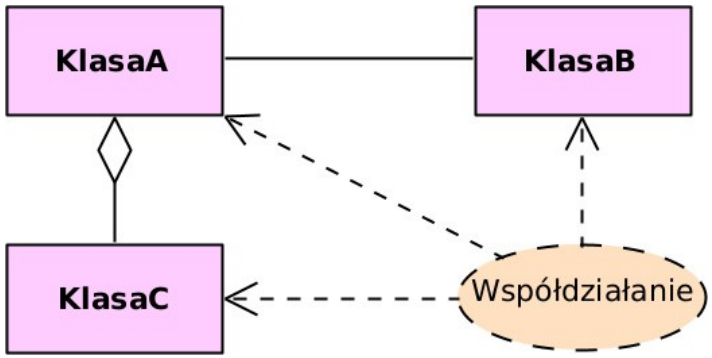


- **Przykład**



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.**

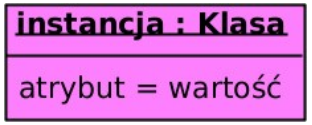


4

Diagram obiektów

Diagram obiektów /object diagram/

- Modeluje instancje klas i relacje między nimi.
- **Obiekt** – instancja klasy:
  - nazwana lub anonimowa (tylko nazwa klasy),
  - kompletna lub częściowa (tylko wybrane atrybuty):
    - atrybuty mają konkretne wartości.



- Nazwa obiektu jest podkreślona:
  - typowa składnia:
    ```
    <instance> ::= [<name>] : [<package>::]*<classifiername>
    ```
  - pełna składnia:
    ```
    <instance> ::= {<name> ['/'<rolename>] | '/'<rolename>}
    [':' [<package>::]*<classifiername> [',' <classifiername>]*]
    ```

<name> – nazwa obiektu;

<rolename> – nazwa roli pełnionej przez obiekt;

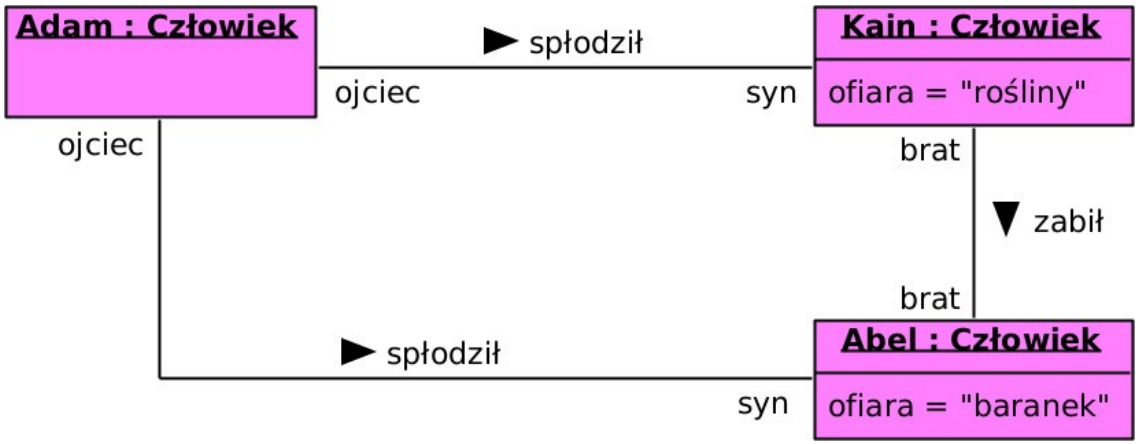
<package> – pakiet;

<classifiername> – nazwa klasy, której obiekt jest instancją.

Diagram obiektów

- Związki między obiektami:
  - zależność,
  - asocjacja 1 do 1.

- **Przykład:**



5

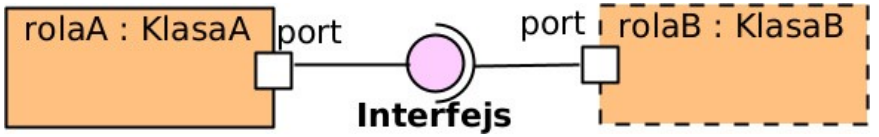
Diagram struktur złożonych

Diagram struktur złożonych
/composite structure diagram/

- Modeluje strukturę złożoną hierarchicznie:
  - wewnętrzną strukturę klasyfikatora (np. klasy),
  - współdziałanie.
- Pakiet grupuje klasy w czasie kompilacji,
  a struktura złożona grupuje klasy w czasie wykonania programu.
- Główne elementy struktury:
  - **część** /part/ struktury (m.in. jako **rola** /role/ klasy),
  - **referencja** do części /referenced part/,
  - **interfejs** (i **port**) dostępu od i do struktury i jej części,
  - **łącznik** /connector/ – relacja między częściami i części z interfejsem,
  - **klasyfikator** (np. klasa),
  - **współdziałanie** /collaboration/ ról w osiąganiu celu struktury,
  - **użycie współdziałania** /collaboration use/.

Wewnętrzna struktura /internal structure/

- Budowa klasyfikatora (np. klasy)
  zawarta w jego przedziale „wewnętrzna struktura”:
  - jego **części**
    (gdy klasyfikator jest w relacji agregacji
    lub kompozycji z częścią),
  - **referencje** do NIE jego części
    (gdy klasyfikator jest w relacji asocjacji z częścią),
  - **role** pełnione przez części,
  - **łączniki** – relacje między nimi i z interfejsami,
  - **liczba** ról i części.
- **Interfejs** modeluje funkcjonalność struktury:
  - bez szczegółów realizacji,
  - w postaci „lizaka” (gdy zapewniany)
    lub „łapki” (gdy wymagany),
  - może być zaczepiony w porcie.



Wewnętrzna struktura

- **Składnia nazwy części i referencji do części:**
  - typowa składnia:
    ```
    <part> ::= [<name>] : [<package>::]*<classifiername>
    ```
  - pełna składnia:
    ```
    <part> ::= {<name> [‘/’<rolename>] | ‘/’<rolename>}
    [‘:’ [<package>::]*<classifiername> [‘,’ <classifiername>]*]
    ```

<name> – nazwa części;

<rolename> – nazwa roli pełnionej przez część;

<package> – pakiet;

<classifiername> – nazwa klasy, której część jest instancją.

- **Składnia nazwy łącznika:**
  ```
  <connector> ::= ( [<name>] ’:’ <associationname> ) |
                  ( [<name>] ’:’ <associationclassname> ) | [<name>]
  ```

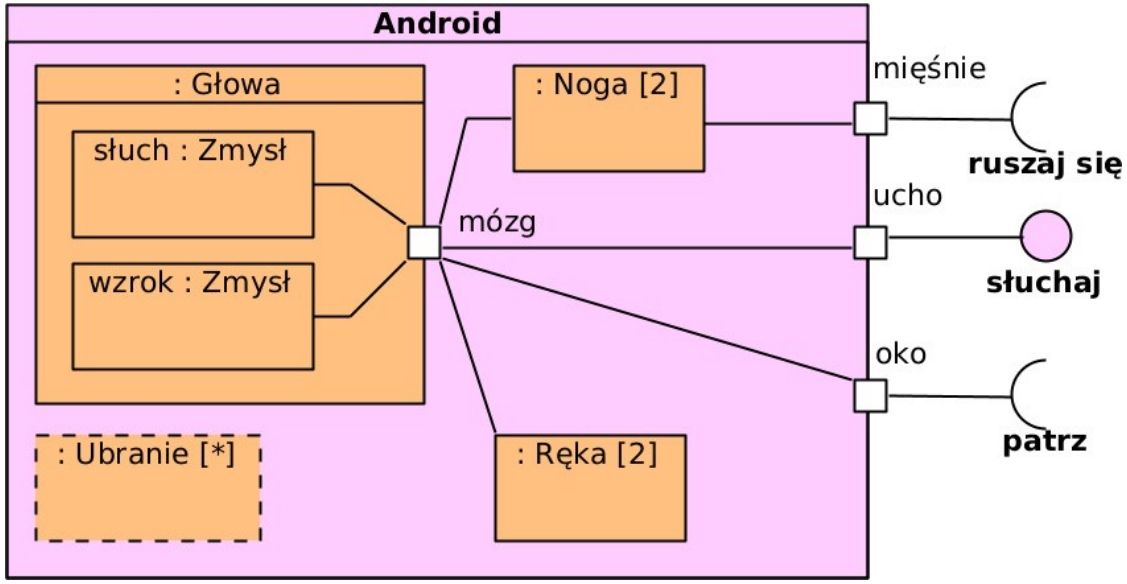
<name> – nazwa łącznika;

<associationname> – nazwa asocjacji, której łącznik jest instancją;

<associationclassname> – nazwa klasy asocjacyjnej, której łącznik jest instancją.

Przykład złożonej wewnętrznej struktury

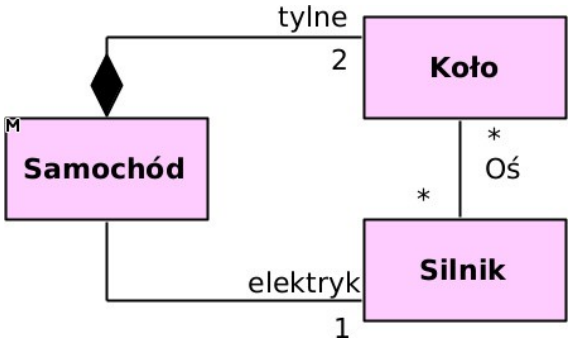
- Części klasyfikatora **Android**:
  Głowa (a w niej: **Zmysły** (**słuch** i **wzrok**)), 2 **Nogi** i 2 **Ręce**.
- **Android** może mieć na sobie (nie w sobie) dowolnie wiele **Ubrań**.
- **Głowa** łączy się z **Nogami**, **Rękami** i otoczeniem **Androida**
  przez port **mózg**.
- **Android** implementuje interfejs (aby nim sterować) **słuchaj** przez port **ucho**.
- **Android** wymaga interfejsów (aby wpływał na swoje otoczenie)
  ruszaj się przez port **mięśnie** i **patrz** przez port **oko**.



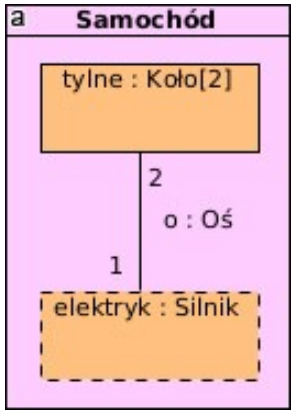
Przykład diagramu wewnętrznej struktury i porównanie z diagramem klas.

na podst. Unified Modeling Language (UML)

- Diagram klas pokazuje:
  - **Samochód** posiada 2 **Koła** w roli **tylne**.
  - **Samochód** związany jest (nie posiada) z 1 **Silnikiem** w roli **elektryk**.
  - Asocjacja **Oś** łączy **Koło** i **Silnik** w stosunku wiele do wielu.

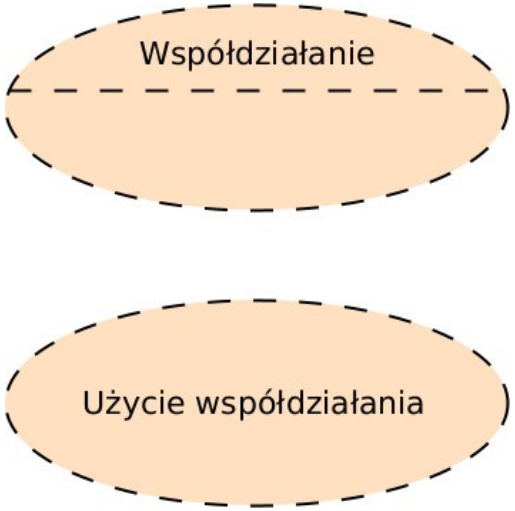


- Diagram wewnętrznej struktury pokazuje (dla konkretnego **Samochodu**):
  - Częściami **Samochodu** są: 2 **Koła** w roli **tylne** i 1 **Silnik** w roli **elektryk**.
  - **Tylne** z **elektrykiem** łączy łącznik **o** (asocjacja **Oś**) w stosunku 2 do 1.



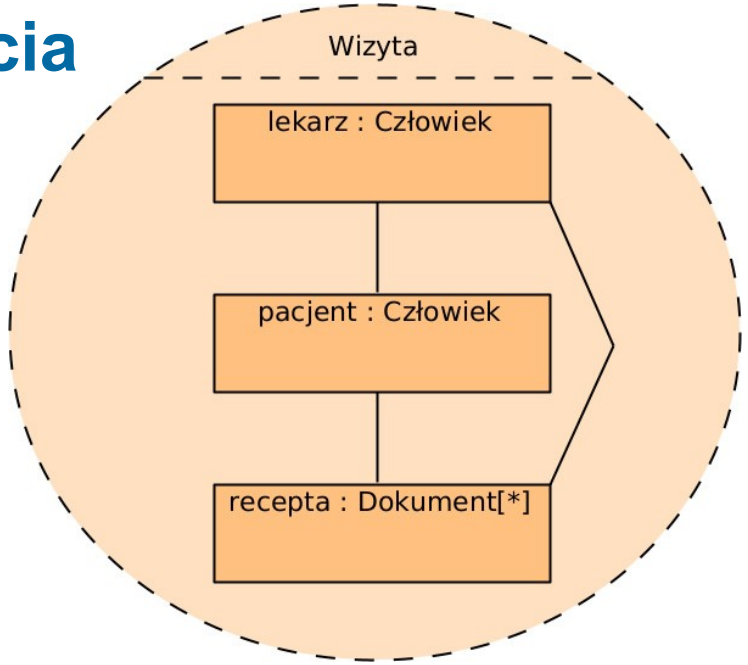
Współdziałanie /collaboration/

- **Modeluje** współdziałanie wybranych ról klas w celu osiągnięcia wspólnych celów (zadań).
- **Zawiera** części (jako **role** klas) i **łączniki** (relacje między nimi):
  - TYLKO te, które są potrzebne do osiągnięcia celów współdziałania;
  - pozostałe (jeśli istnieją) nie są pokazane.
- Klasa może pełnić różne role i uczestniczyć w wielu współdziałaniach.
- **Reprezentacja:** owal z przerywaną krawędzią, nazwą i wewnętrzną strukturą (lub zwykły klasyfikator ze stereotypem «*collaboration*»).
- **Użycie współdziałania** (owal z przerywaną krawędzią i nazwą):
  - w relacji asocjacji z klasami pełniącymi role we współdziałaniu (na diagramie struktur złożonych),
  - w relacji zależności z klasami pełniącymi role we współdziałaniu (na diagramie klas).



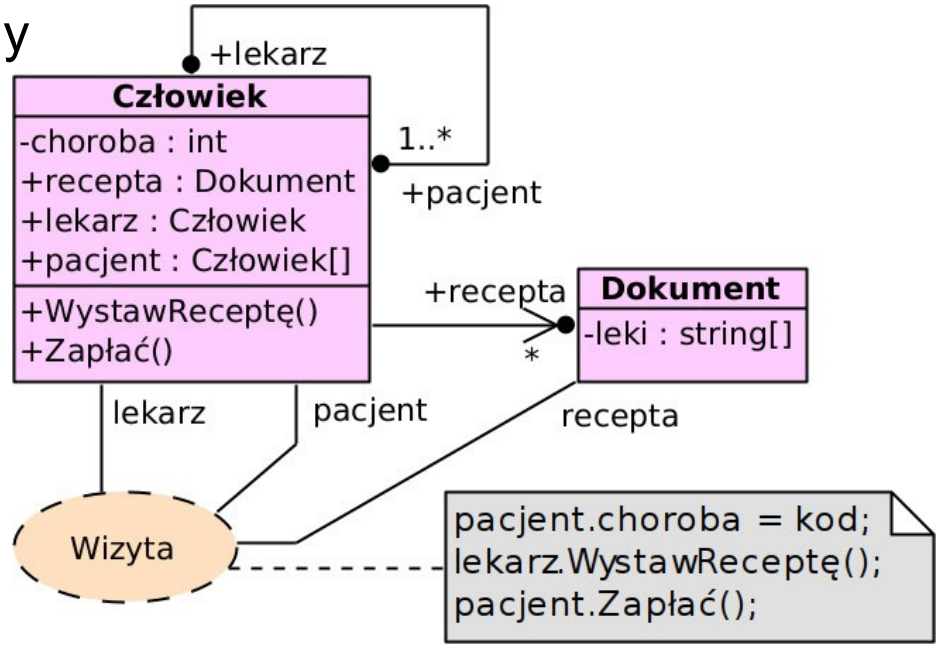
Przykład współdziałania i jego użycia

Diagram konceptualny współdziałania **Wizyta** →



- Diagram klas pokazuje:
  - **Człowiek** w roli **lekarza** jest związany z co najmniej 1 **Człowiekiem** w roli **pacjenta**.
  - to ogólna relacja.
- Współdziałanie **Wizyta** pokazuje:
  - **Człowiek** w roli **lekarza** jest związany z dokładnie 1 **Człowiekiem** w roli **pacjenta**,
  - to szczególna relacja użyta do wykonania zadań Wizyty.

Diagram implementacyjny użycia współdziałania **Wizyta** →



6

Diagram komponentów

Diagram komponentów /component diagram/

- Modeluje strukturę oprogramowania na wyższym poziomie abstrakcji, niż diagram klas.
  - **System składa się z komponentów:**
    - są niezależne od siebie (niezależnie kompilowane i niezależnie implementowane),
    - można je rozbudowywać,
    - można je wymieniać na inne,
    - można je używać w różnych systemach,
    - udostępniają swoją funkcjonalność przez interfejsy i porty.

Diagram komponentów

Komponent /component/

- Klasyfikator ze stereotypem **«*component*»** lub ikoną komponentu.
- **Logiczna część systemu:**
  - implementacyjnie niezależna od innych,
  - rozbudowywalna,
  - wymienialna,
  - ponownie używalna.

- **Udostępnia swoją funkcjonalność** przez interfejsy i porty.
- Może pokazywać szczegóły swej implementacji.
- **Modeluje** np. aplikację, bibliotekę, współdziałanie.
  - Sposób jego modelowania, ulepszania i wdrażania określa cykl życia oprogramowania.
- **Artefakt** /artifact/ – klasyfikator implementujący komponent (np. plik *.jar*).
  - jego wdrożenie i aktualizacja powinny być niezależne.

Zawartość komponentu

- **Stereotypy** («*component*» i dodatkowe inne) – modyfikacja komponentu.
- **Nazwa** – nazwa komponentu.
- Przedziały:
  - **atrybuty** /propeties/ – prywatne atrybuty (własności) komponentu;
  - **interfejsy zapewniane** /provided interfaces/ przez komponent;
  - **interfejsy wymagane** /required interfaces/ przez komponent;
  - **realizacje** /realizations/ – składowe klasyfikatory komponentu realizujące interfejsy;
  - **artefakty** /artifacts/ – klasyfikatory implementujące komponent;
  - **wewnętrzna struktura** /internal structure/ – składowe komponenty (i relacje między nimi) realizujące funkcjonalność komponentu → diagram komponentów;
  - **elementy spakowane** /packaged elements/ – składowe klasyfikatory (klasy, obiekty i ich relacje) realizujące funkcjonalność komponentu → diagram struktur złożonych.

Dodatkowe stereotypy komponentu

- Określają, co lub jak komponent modeluje, np.:
  - **«*document*»** – dokument dowolnego typu;
  - **«*file*»** – plik z danymi lub kodem źródłowym;
  - **«*library*»** – biblioteka programu;
  - **«*executable*»** – wykonywalny program;
  - **«*process*»** – proces (działa etapowo - stanowo);
  - **«*service*»** – usługa (działa bezetapowo - bezstanowo);
  - **«*subsystem*»** – podsystem dużej skali (złożony z wielu komponentów);
  - **«*entity*»** – encja (trwałe dane), zwykle bez funkcjonalności;
  - **«*table*»** – tabela bazy danych;
  - **«*specification*»** – specyfikacja komponentu (są interfejsy, NIE ma implementacji);
  - **«*realization*»** – konkretna realizacja komponentu «*specification*».

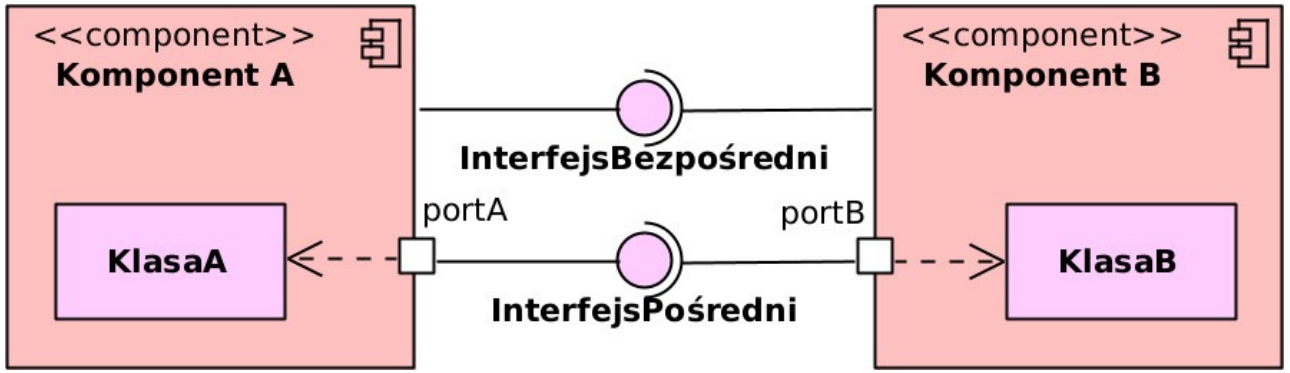
Relacje między komponentami

- Komponenty mogą być powiązane ze sobą:
  - **pośrednio** – łączy je interfejs (np. z użyciem portów),
  - **bezpośrednio** – łączy je inna relacja (np. z użyciem portów),
  - **hierarchicznie** – jeden jest częścią drugiego.

- Komponent używający porty to opakowany klasyfikator.

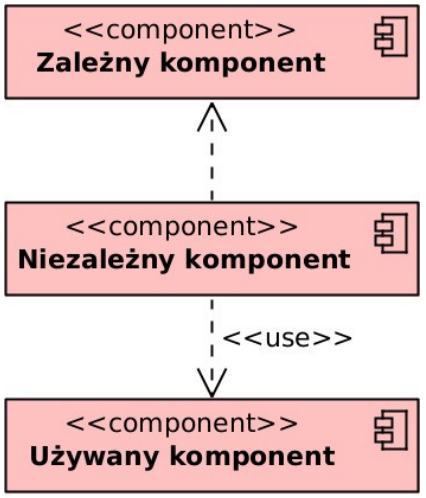
Interfejsy

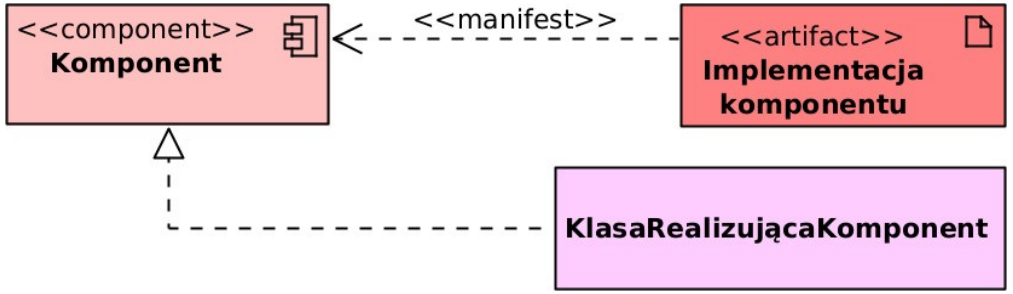
- Dostęp komponentu do innego komponentu
  - za pośrednictwem interfejsu:
    - w postaci „lizaka” (gdy zapewniany) lub „łapki” (gdy wymagany) lub w postaci klasy ze stereotypem: «*interface*»,
    - interfejs może być zaczepiony w porcie.
- **Relacja zależności** – łączy spakowane elementy (klasyfikatory), które zapewniają interfejsy lub wymagają interfejsów komponentu, z portami tych interfejsów:
  - wskazuje na klasyfikator.
- Diagramy klas, modelujące komponent, powinny zawierać jego interfejsy i spakowane klasy.

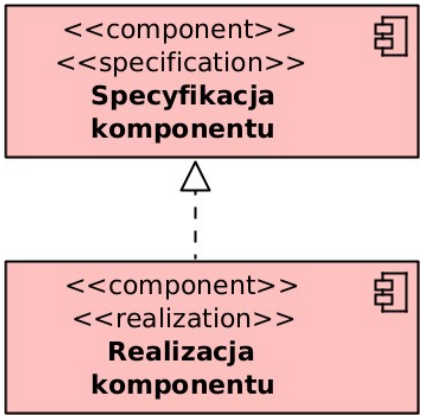


Relacje zależności i realizacji

- Dostęp komponentu do innego komponentu
  - BEZ pośrednictwa interfejsu;
  - przez **relację zależności**:
    - gdy komponent (lub artefakt) jest zależny od innego komponentu,
    - rodzaj zależności może określić jej stereotyp.
  - przez **relację realizacji**:
    - gdy komponent (lub klasa) jest realizacją innego komponentu.

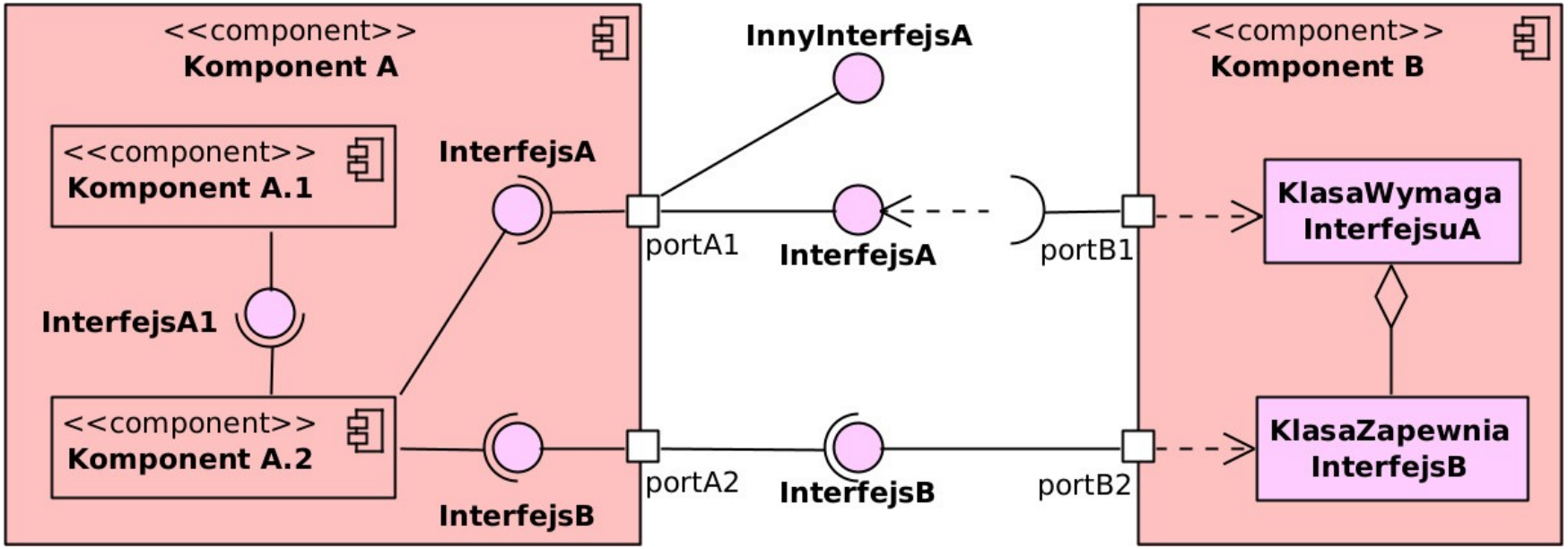






Przykład relacji między komponentami (i spakowanymi elementami)

- **Komponent A** pokazuje swoją wewnętrzną strukturę:
  - **Komponent A.1** współpracuje z **Komponentem A.2** przez **InterfejsA1**.
- **Komponent B** pokazuje swoje spakowane elementy:
  - klasy uczestniczące we współpracy z **Komponentem A** i ich związek.
- **portA1** udostępnia 2 interfejsy.



7

Diagram wdrożenia

Diagram wdrożenia /deployment diagram/

- Modeluje fizyczną i logiczną strukturę systemu:
  - powiązanie oprogramowania ze sprzętem (na którym jest wdrażane), innym systemem (z którym współpracuje) i artefaktami;
  - jako układ węzłów powiązanych strukturalne i komunikacyjne:
    - fizyczne urządzenia,
    - środowiska wykonawcze,
    - artefakty (elementy fizyczne),
    - komponenty (elementy logiczne).
- W fazie specyfikacji wymagań: umiejscawia logiczne komponenty oprogramowania w infrastrukturze klienta.
- W fazie wdrożenia: dokumentuje sposób instalacji oprogramowania.
- Głównie dla systemów wbudowanych i rozproszonych typu klient-serwer.
- Klasyfikator ***«deployment spec»*** – plik ze specyfikacją wdrożenia.

Diagram wdrożenia

Węzeł /node/

- Fizyczny element systemu lub jego otoczenia: sprzęt lub oprogramowanie.
- Węzeł może mieć stereotyp – określa jego rodzaj:
  - **«*device*»** – fizyczne urządzenie (sprzęt),
  - **«*executionEnvironment*»** – środowisko uruchomieniowe lub wykonawcze komponentów systemu.
  - Można definiować własne stereotypy.

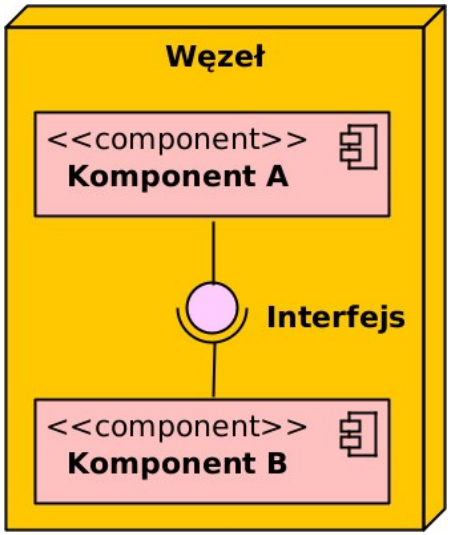
- Węzeł może być zagnieżdżony.
- **Elementy związane z węzłem** (węzły, komponenty, artefakty):
  - są w nim umieszczone,
  - są z nim w relacji bezpośredniej (zależność i asocjacja),
  - są z nim w relacji pośredniej (np. interfejs łączący komponenty).
- Relacja asocjacji między węzłami to ścieżka komunikacji między nimi.

Diagram wdrożenia

Komponent /component/

- Klasyfikator ze stereotypem **«*component*»** lub ikoną komponentu.
- **Logiczny element:**
  - część oprogramowania systemu,
  - implementacyjnie niezależny od innych.

- **Relacje między komponentami:**
  - zależność, asocjacja, agregacji, kompozycji, uogólnienie;
  - powiązanie przez interfejs (w postaci „lizaka” i „łapki”).
- Komponent może być zagnieżdżony.
- **Wdrożenie komponentu** przez węzeł:
  - umieszczenie go w węźle.



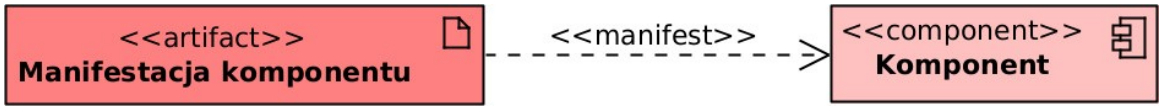
Artefakt /artifact/

- Klasyfikator ze stereotypem **«*artifact*»** lub ikoną dokumentu.
- **Fizyczny element:**
  - używany lub wytwarzany przez wdrażany system;
  - manifestacja komponentu systemu (jego fizyczna reprezentacja, implementacja);
  - np. informacja, plik źródłowy, plik biblioteki, plik wykonywalny, plik danych, tablica bazy danych.
- **Relacje między artefaktami**:
  - zależność, asocjacja, agregacji, kompozycji, uogólnienie.
- Artefakt może być zagnieżdżony.
- Artefakt może mieć atrybuty.

Artefakt

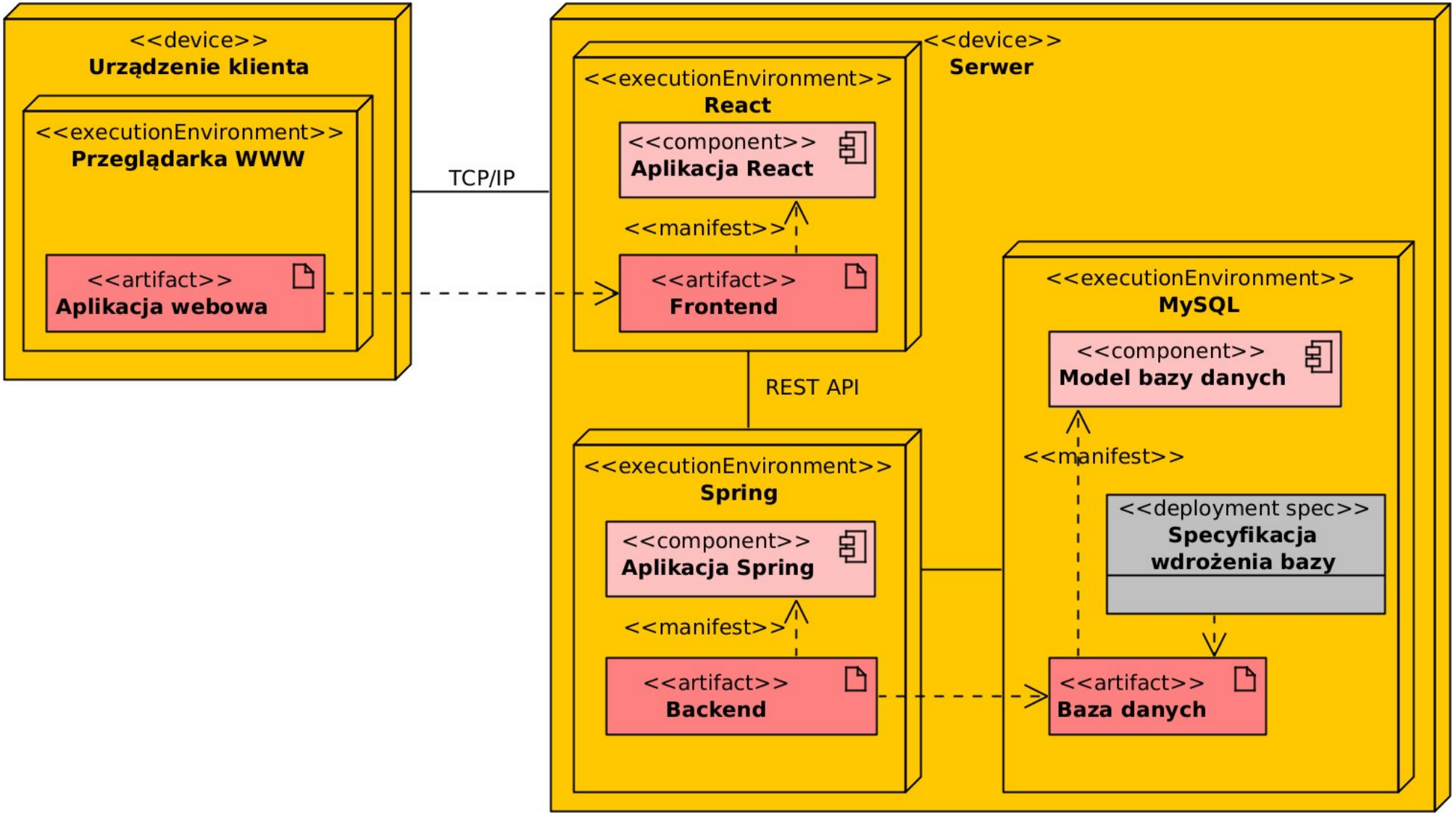
- **Używanie lub wytwarzanie artefaktu** przez węzeł:
  - umieszczenie go w węźle,
  - połączenie go z węzłem relacją zależności **«*deploy*»**.

- **Manifestacja komponentu przez artefakt**:
  - połączenie go z komponentem relacją zależności **«*manifest*»**.



Przykład diagramu wdrożenia

- Koncepcja wdrożenia webowej aplikacji bazodanowej.



8

Diagram profilu

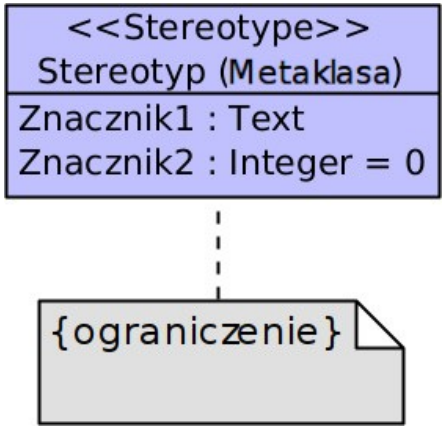
Diagram profilu /profile diagram/

- Tworzy nowy metamodel UML (model semantyki i składni UML).
- Rozszerza lub modyfikuje istniejący metamodel UML
  - nie narusza jego semantyki,
  - nie zmienia meta-metamodelu (MOF).
- Dostosowuje metamodel UML do innego zastosowania:
  - dla różnych platform, np.: J2EE, .NET;
  - dla różnych domen, np.: architektura czasu rzeczywistego, architektura zorientowaną na usługi (SOA).
  - zmienia interpretację modelu, a NIE model.

Więcej o metamodelu UML w prezentacji „Definiowanie metamodelu”.

Diagram profilu

- **Profil** /profile/ – zbiór rozszerzeń metamodelu UML.
- **Metaklasa** /metaclass/ – rozszerzany element metamodelu.
- Elementy rozszerzenia metaklasy przez profil:
  - **stereotyp** /stereotype/
    – typ rozszerzenia,
  - **znacznik** /tagged value/
    – dodatkowy parametr rozszerzenia,
  - **ograniczenie** /constraint/
    – warunek do spełnienia przez rozszerzenie.

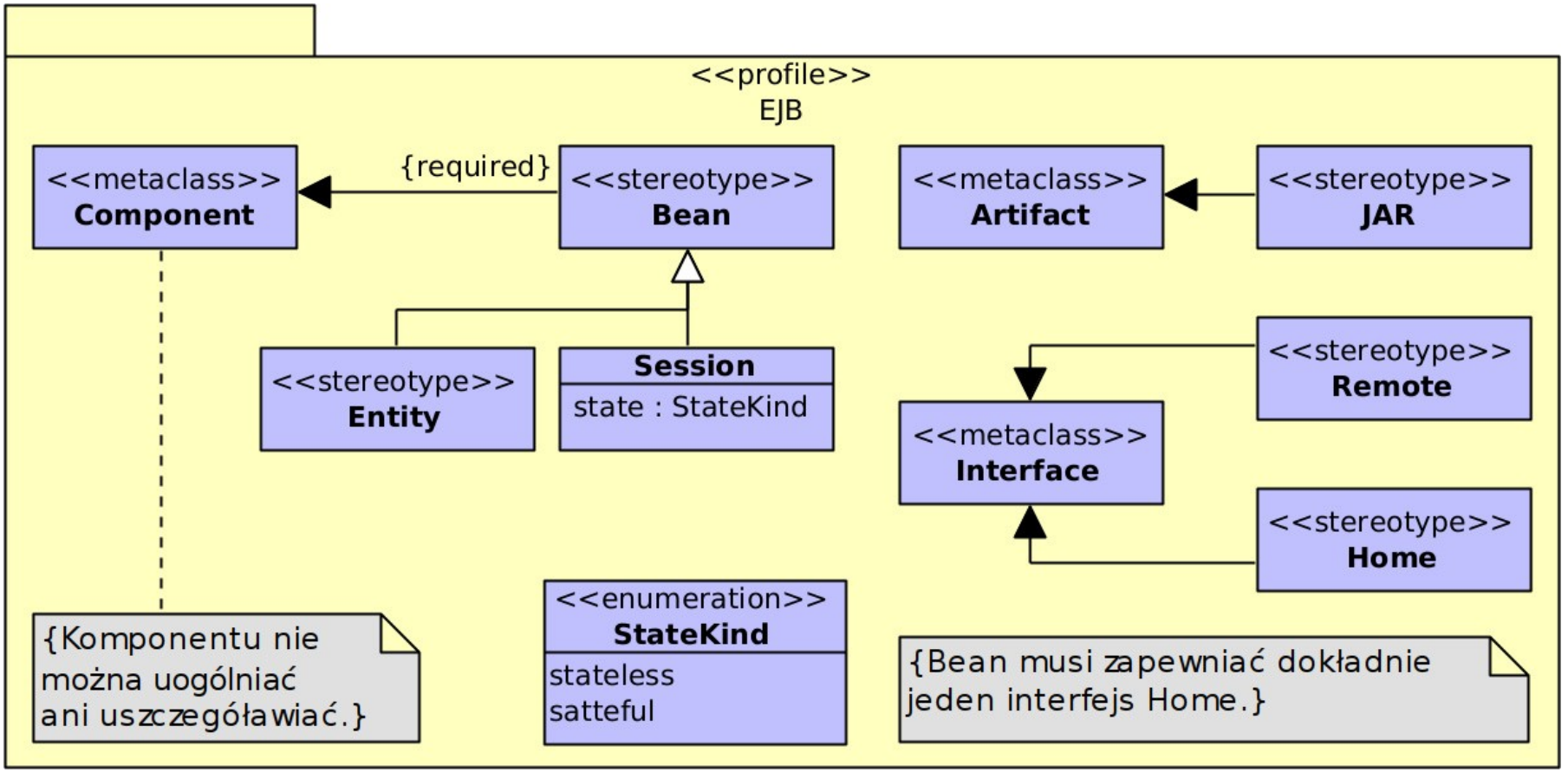


- Profil stosuje się do konkretnych elementów modelu, np. klas, atrybutów, operacji, czynności.
- **Relacja rozszerzenia** /extension/ (ciągła linia z grotem ▲).
  - Grot ▲ wskazuje rozszerzaną metaklasę.
- **Pakiet ze stereotypem «*profile*»** zawiera definicję profilu.

Przykład diagramu profilu

na podst. Unified Modeling Language (UML)

- Stereotyp **Bean** jest wymagany dla metaklasy **Component**.
- Instancja jego podklasy **Entity** lub **Session** musi być dołączona do każdej instancji **Component**.



Temat następnej prezentacji

Język UML – modelowanie systemu informatycznego