

# Projektowanie obiektowe oprogramowania

## Wykład 2 - UML

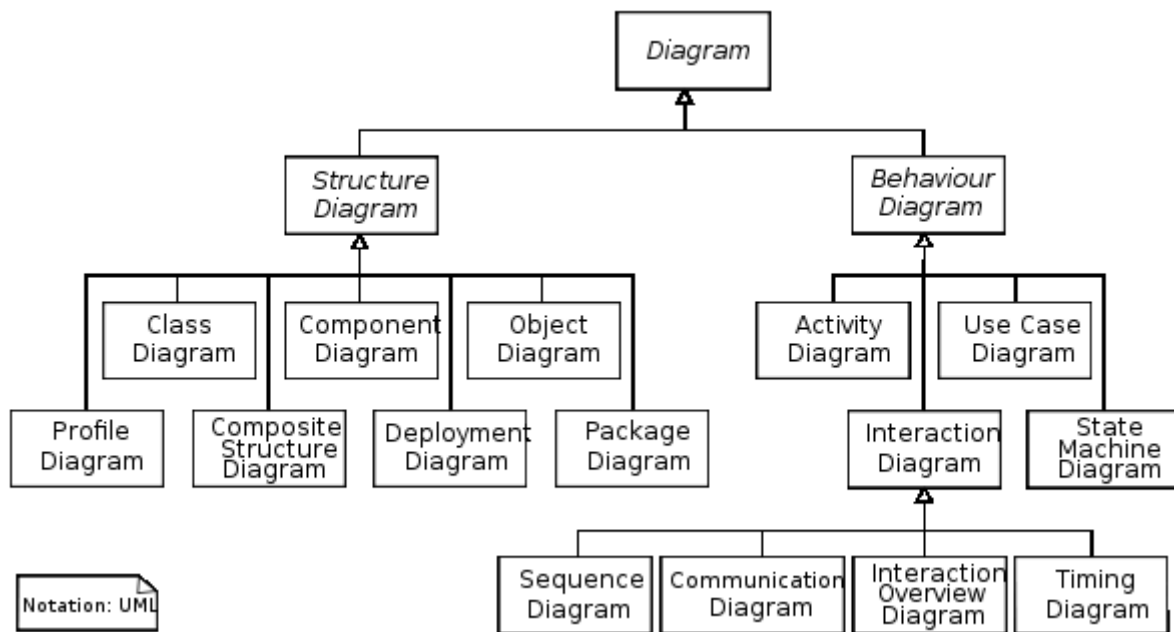
### Wiktor Zychla 2024

---

#### Spis treści

1	Wprowadzenie .....	2
2	Diagramy klas .....	3
2.1	Hierarchia modeli .....	3
2.1.1	Diagram modelu pojęciowego .....	3
2.1.2	Diagram modelu obiektowego (diagram klas).....	4
2.1.3	Diagram modelu implementacyjnego (relacyjnego) .....	6
2.2	Jeszcze o formalizmie diagramów klas - klasy i asocjacje .....	8
2.3	Składowe .....	9
2.4	Dziedziczenie .....	10
3	Diagramy obiektów .....	11
4	Diagramy stanów.....	12
5	Diagramy czynności .....	14
6	Diagramy sekwencji .....	16
7	Diagramy komponentów.....	18
8	Literatura .....	19

# 1 Wprowadzenie



Dwie rodziny diagramów - diagramy **struktur** i diagramy **zachowań** (dynamiki):

- Diagramy struktur – służą do dokumentowania statycznych elementów systemu i relacji/powiązań między nimi
- Diagramy zachowań - służą do dokumentowania dynamicznych elementów systemu np. procesów/algorytmów/przypadków użycia

Jest wiele narzędzi wspierających projektowanie diagramów, począwszy od lekkich, przeglądarkowych typu [draw.io](https://draw.io), [yuml.me](https://yuml.me) czy [umletino](https://umletino.com) po pełnoprawne narzędzia wizualne, z których na uwagę zasługują dwa:

- [Enterprise Architect](https://www.enterprise-architect.com/) – narzędzie płatne
- [Visual Paradigm](https://visual-paradigm.com/) - posiada wersję [Community](https://visual-paradigm.com/community/) bezpłatną do zastosowań niekomercyjnych

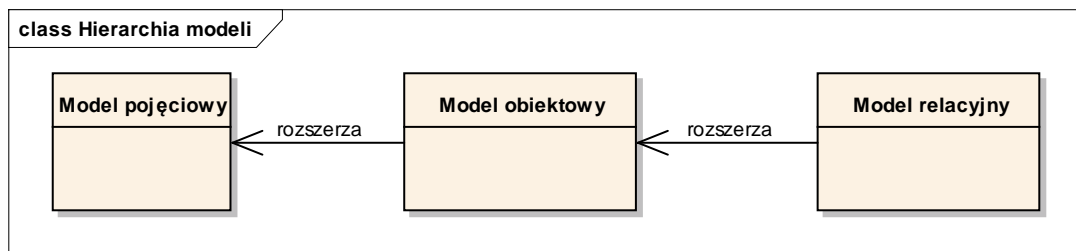
## 2 Diagramy klas

**Diagramy klas (Class Diagram)** służą do reprezentacji m.in.

- Modeli pojęciowych
- Zbiorów klas (jakiegoś języka obiektowego)
- Schematów baz danych

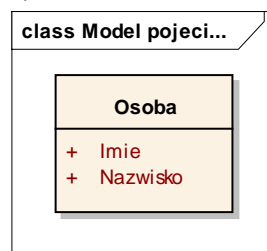
### 2.1 Hierarchia modeli

Formalnie w UML występuje Diagram klas (Class Diagram). Ale ten sam formalizm służy do reprezentacji **trzech** różnych typów diagramów, z których każdy występuje w innej fazie projektowania.

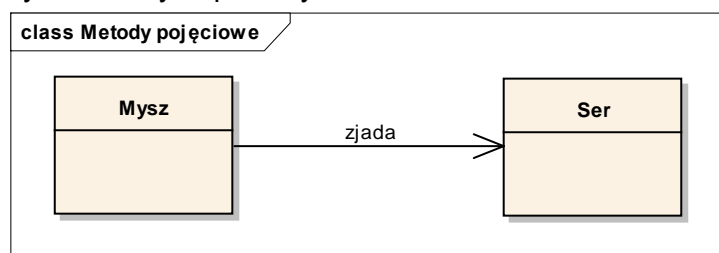


#### 2.1.1 Diagram modelu pojęciowego

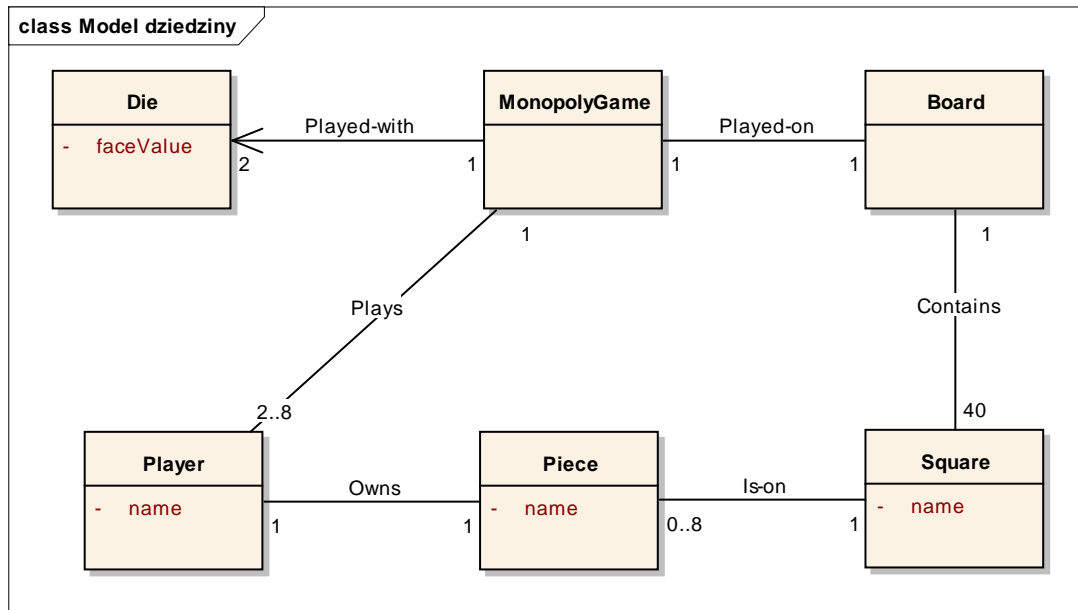
- Jest elementem projektu analitycznego
- Służy ustaleniu wspólnego języka w projekcie
- Służy weryfikacji podstawowych elementów struktury dziedziny projektu
- Pojęcia i atrybuty (tylko publiczne)



- Asocjacje (relacje) między pojęciami („ma”, „używa”, „płaci się za pomocą”)
- Asocjacje mogą być skierowane, wtedy kierunek strzałki i jej etykieta powinien być tak dobrany żeby można było „przeczytać” zdanie



- Brak dziedziczenia i innych ograniczeń specyficznych dla struktury stricte obiektowej
- Brak metod (!)

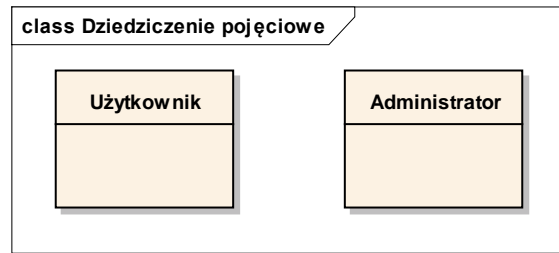


### 2.1.2 Diagram modelu obiektowego (diagram klas)

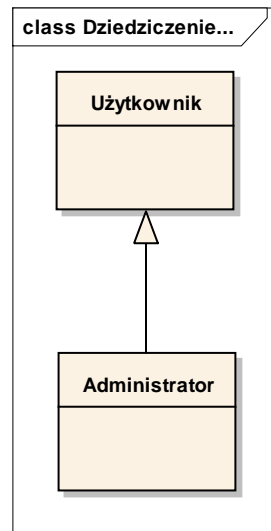
- Jest elementem projektu architektury
- Punktem wyjścia jest diagram modelu pojęciowego
- Na etapie projektowania obiektowego należy refaktoryzować model pojęciowy do stanu, w którym pojęcia reprezentowane są przez klasy (w rozumieniu obiektowym)
- Refaktoryzacja polega na:
  - **Usuwanie** zbędnych pojęć, które są reprezentowane przez jedną i tę samą klasę (na przykład pojęcia Użytkownik i Administrator staną się jedną i tą samą klasą)
  - **Dodawanie** nowych klas (na przykład tam gdzie do reprezentacji relacji potrzebna jest pomocnicza klasa)
  - **Rozróżnianiu** atrybutów publicznych, prywatnych, statycznych itd.
  - **Wprowadzaniu** metod do interfejsów klas
  - **Zamienianiu** wszystkich relacji z diagramu modelu pojęciowego na relacje występujące w świecie obiektowym:
    - asocjacja,
    - agregacja,
    - kompozycja,
    - dziedziczenie,
    - implementowanie interfejsu
    - metoda obiektu (przyjmująca parametr lub zwracająca wartość)

Po tej fazie zamiany relacji, na diagramie klas nie może pozostać żadna asocjacja, której nie da się zaimplementować w języku obiektowym

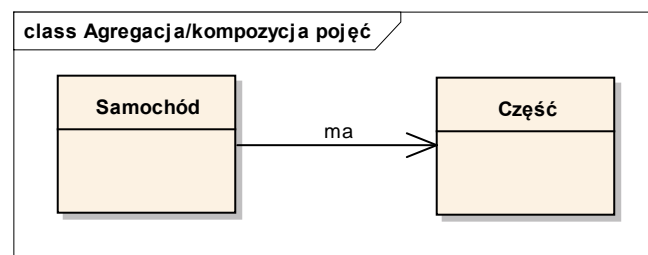
Przykład 1:



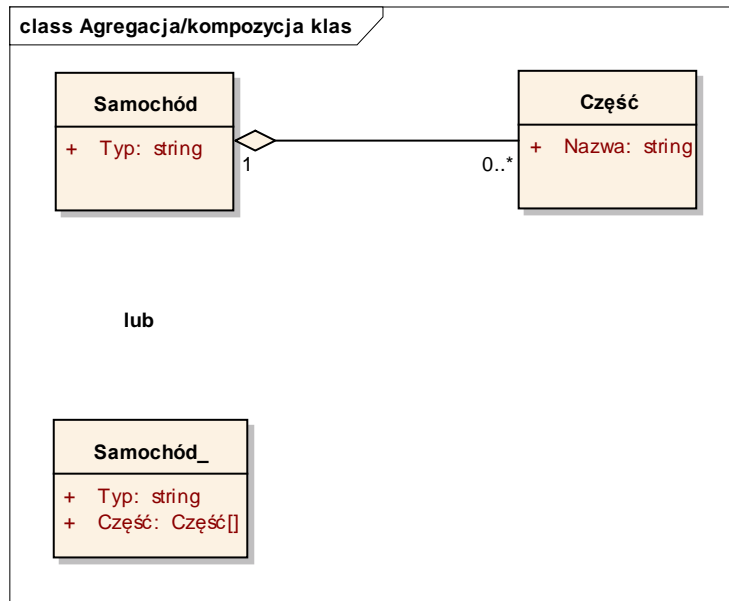
VS



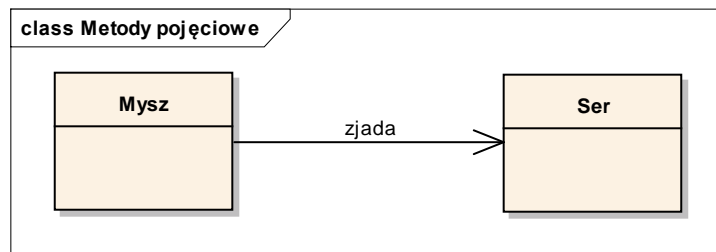
Przykład 2:



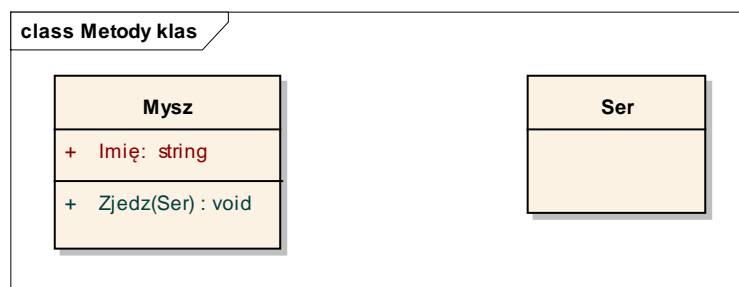
VS



Przykład 3:



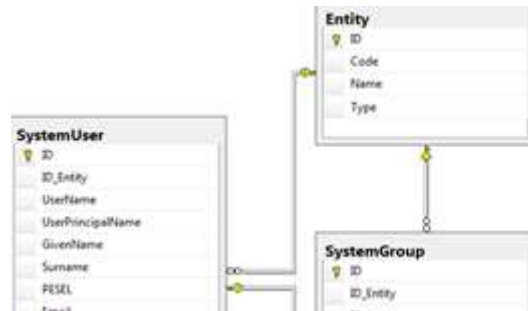
VS



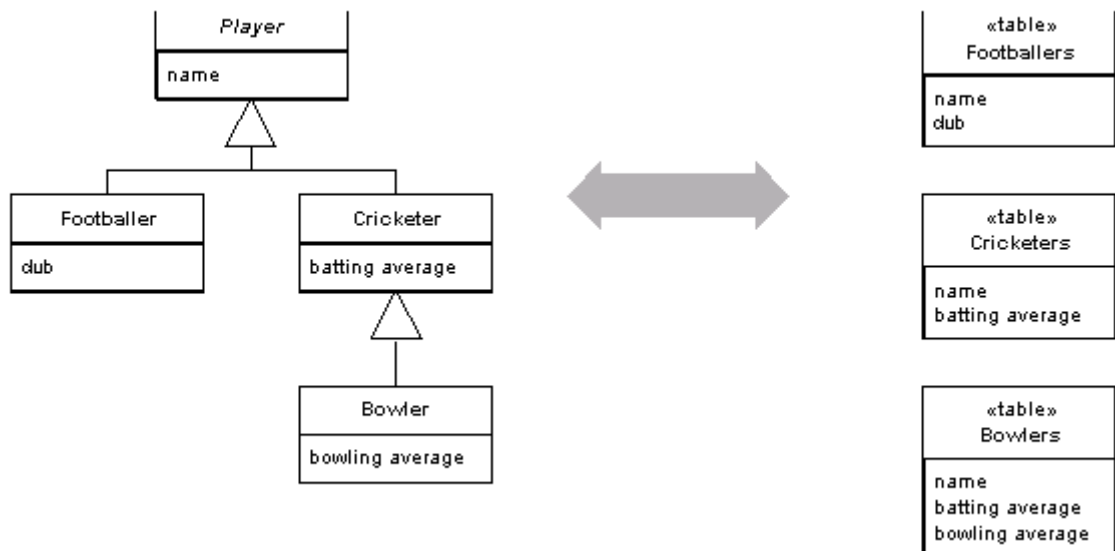
### 2.1.3 Diagram modelu implementacyjnego (relacyjnego)

- To diagram reprezentujący strukturę relacyjnej bazy danych
- Reprezentuje fizyczną strukturę właściwą do utrwalania obiektów
- Punktem wyjścia jest diagram klas
- Podczas refaktoryzacji usuwa się z diagramu klas wszystkie te relacje, których nie da się reprezentować w świecie relacyjnym
  - Nie ma metod
  - Nie ma dziedziczenia, zamiast tego wybiera się jeden ze sposobów implementacji dziedziczenia
    - Table-per-concrete-type
    - Table-per-hierarchy

- Table-per-type
  - Wprowadza się sztuczne identyfikatory główne (ID) (wzorec tzw. Surrogate key, [https://en.wikipedia.org/wiki/Surrogate\\_key](https://en.wikipedia.org/wiki/Surrogate_key))
  - Nie ma relacji wiele-wiele, zamiast tego są pomocnicze tabele do reprezentacji takich relacji

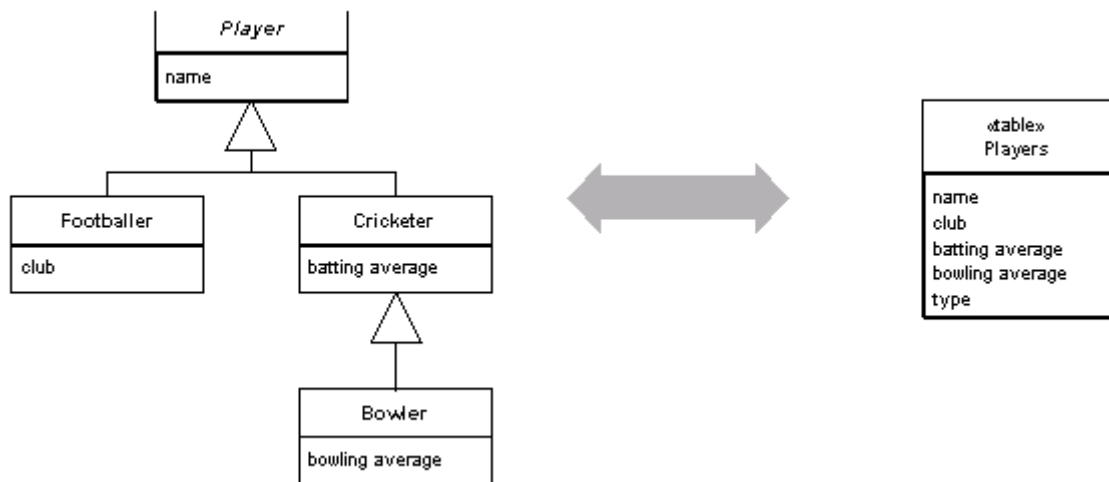


### 2.1.3.1 Table per concrete type



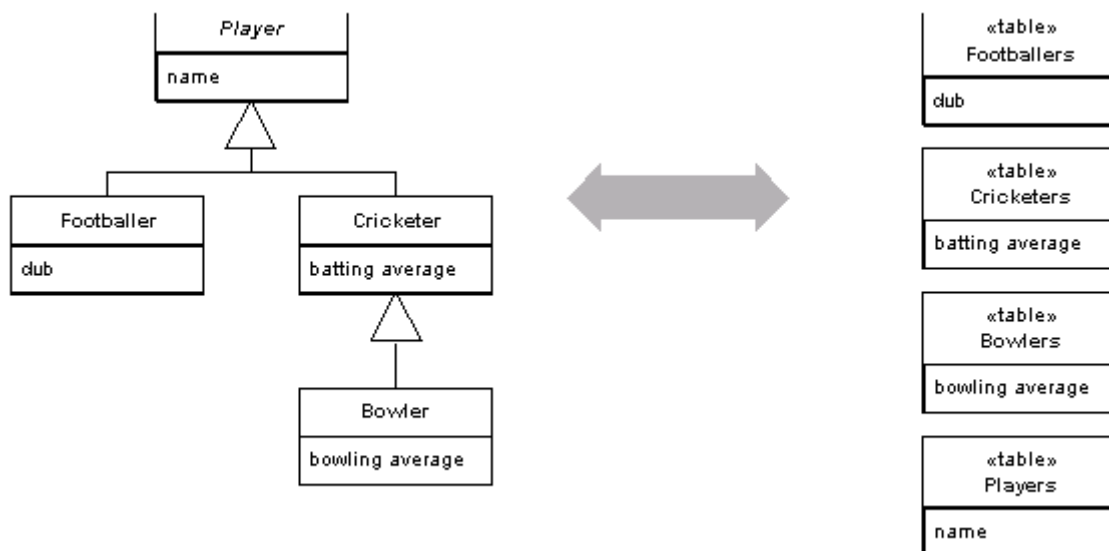
Hierarchia obiektowa zamodelowana jako osobne tabele dla każdej z konkretnych klas.

### 2.1.3.2 Table per hierarchy



Hierarchia obiektowa zamodelowana jako jedna tabela z dodatkową kolumną dyskryminatora.

### 2.1.3.3 Table per type

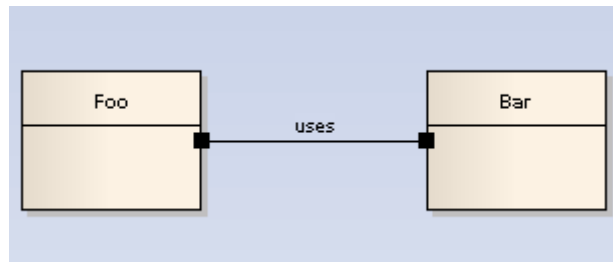


Hierarchia obiektowa zamodelowana jako osobne tabele, **w tym** tabela dla klasy bazowej oraz tabele dla każdej klasy potomnej, z relacjami

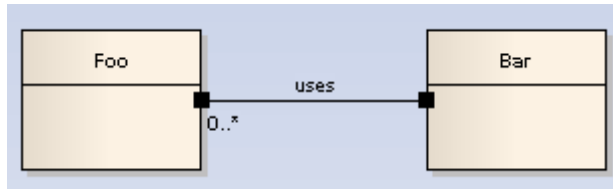
## 2.2 Jeszcze o formalizmie diagramów klas - klasy i asocjacje

- Zależności (strzałka przerywana) – brak informacji o rodzaju zależności, może być:
  - Tworzy
  - Wykorzystuje (zmienna lokalna)
  - Wykorzystuje (parametr metody)
  - Nadklasa lub interfejs
- Nazwy asocjacji

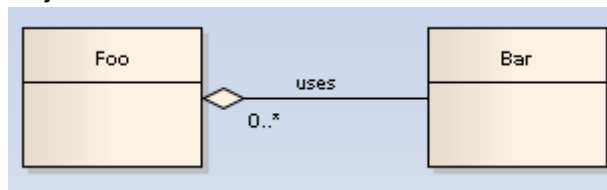




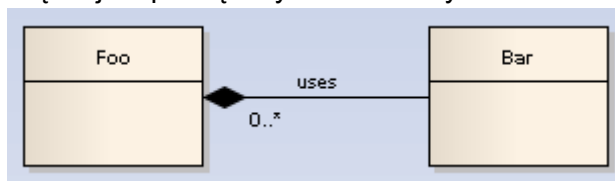
- Liczebność : 1, 1..\*, 0..1, \*, 0..\*, n, 1..n, 0..n, n..m, n..\*



- Agregacja vs kompozycja
  - Agregacja – luźniejsza

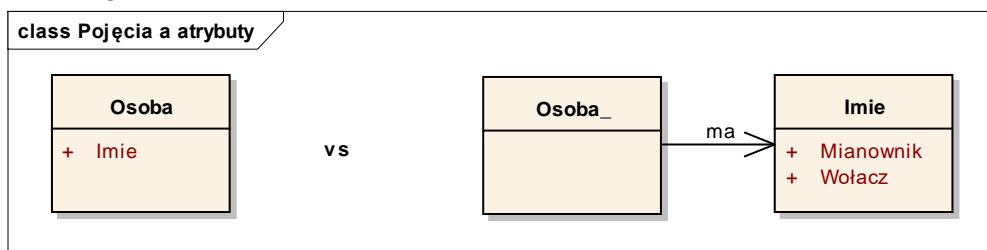


- Kompozycja - ściślejsza
  - Instancja reprezentująca część może należeć tylko do jednej instancji złożonej
  - Czas życia części jest powiązany z czasem życia całości

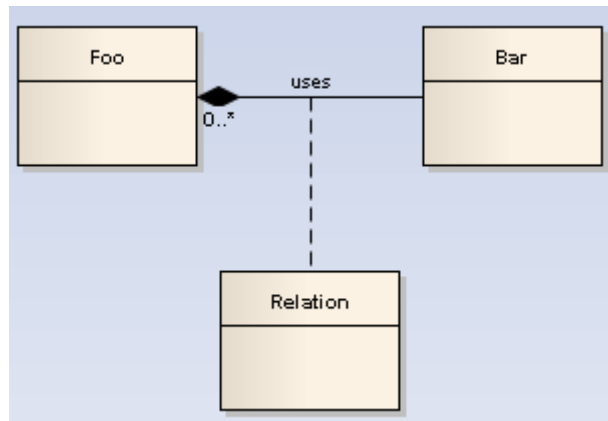


## 2.3 Składowe

- Składowa prywatna, publiczna, chroniona, stała, statyczna, kolekcja, atrybut pochodny
- Metoda prywatna, publiczna, chroniona, internal, abstrakcyjna, statyczna, konstruktor, parametry
- Atrybut wpisany vs asocjacja – kiedy używać? Atrybut: typ prosty, asocjacja do typu złożonego

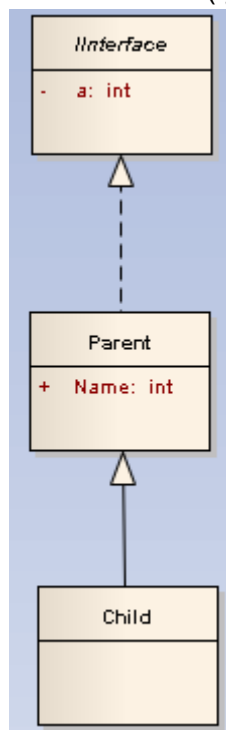


- Klasa asocjacyjna – do modelowania relacji wiele-wiele



## 2.4 Dziedziczenie

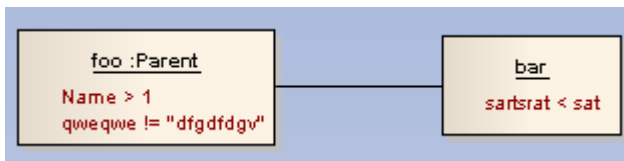
- Realizacja – implementacja interfejsu
  - Generalizacja, specjalizacja – dziedziczenie (tylko w zależności od kierunku)



### 3 Diagramy obiektów

**Diagramy obiektów (Object Diagram)** służą do reprezentacji instancji obiektów i ich wewnętrznego stanu

- Migawka stanu systemu – obiekt określonego typu w określonym stanie w pewnym momencie swojego życia
- Enterprise Architect: Advanced / Instance Classifier – umożliwia wybór typu dla instancji
- EA: Advanced / Set Run State – umożliwia określenie stanu obiektu



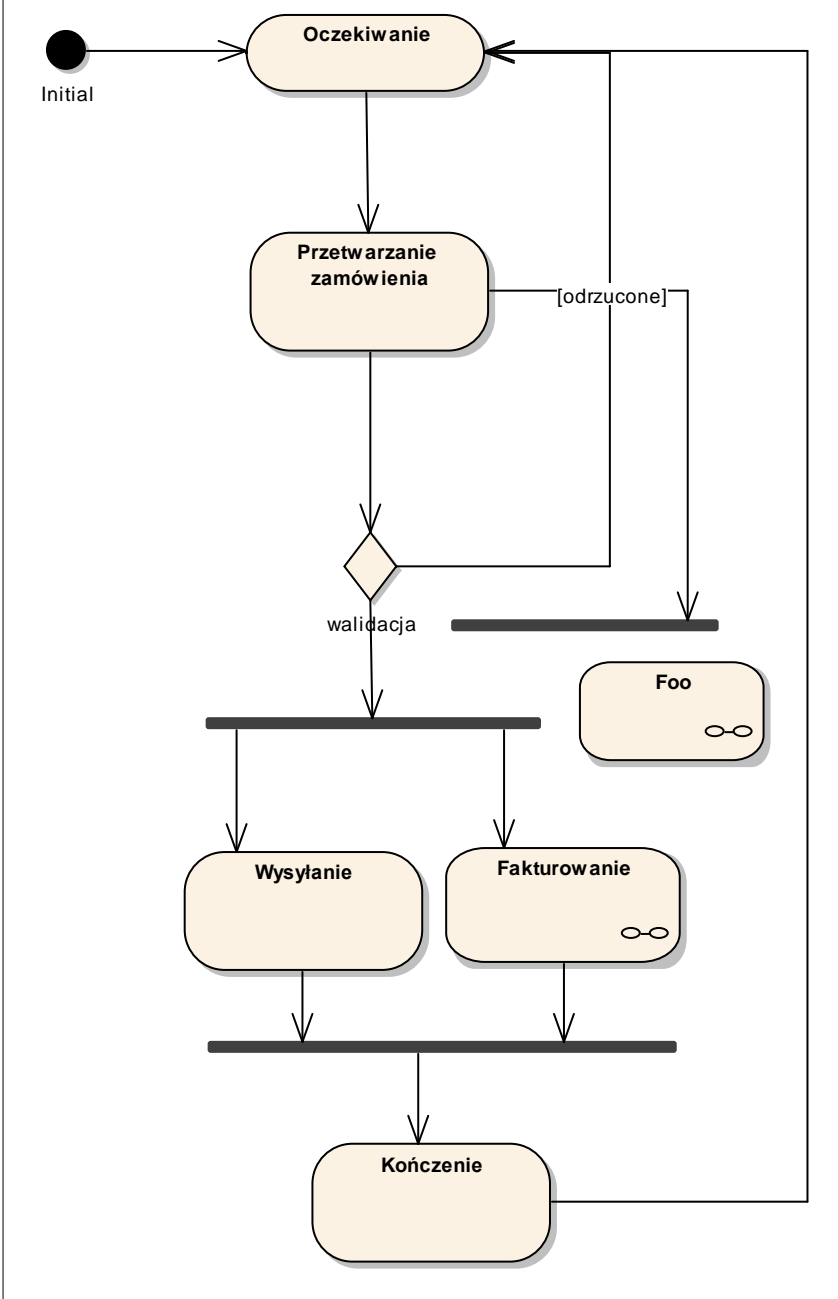
- Visual Paradigm: osobny typ diagramu – Object diagram, na diagramie można dodać instancję obiektu, wskazać jej typ (Add Classifier/Select classifier) spośród wcześniej zdefiniowanych typów, a następnie w specyfikacji (Open specification) określać „sloty” czyli zawartość pamięci obiektu

## 4 Diagramy stanów

**Diagramy stanów (State Diagram, State Machine)** służą do reprezentacji maszyn stanowych

- Stany i przejścia (akcje) – stany to bloczki, a akcje to strzałki
- Stany – nazwane rzeczownikowo/przymiotnikowo (oczekiwanie/przetwarzanie, oczekujący/aktywny/przydzielony)
- Akcje – nie nazywają się
- Przykładowy schemat (rysunek poniżej)
  - Stany – oczekiwanie, przetwarzanie
  - Wariant – nazwany
  - Zrównoleganie – wysyłanie, fakturowanie
  - Stan kompozytowy (pojedynczy stan jest osobnym diagramem)

stm Diagram stanów



## 5 Diagramy czynności

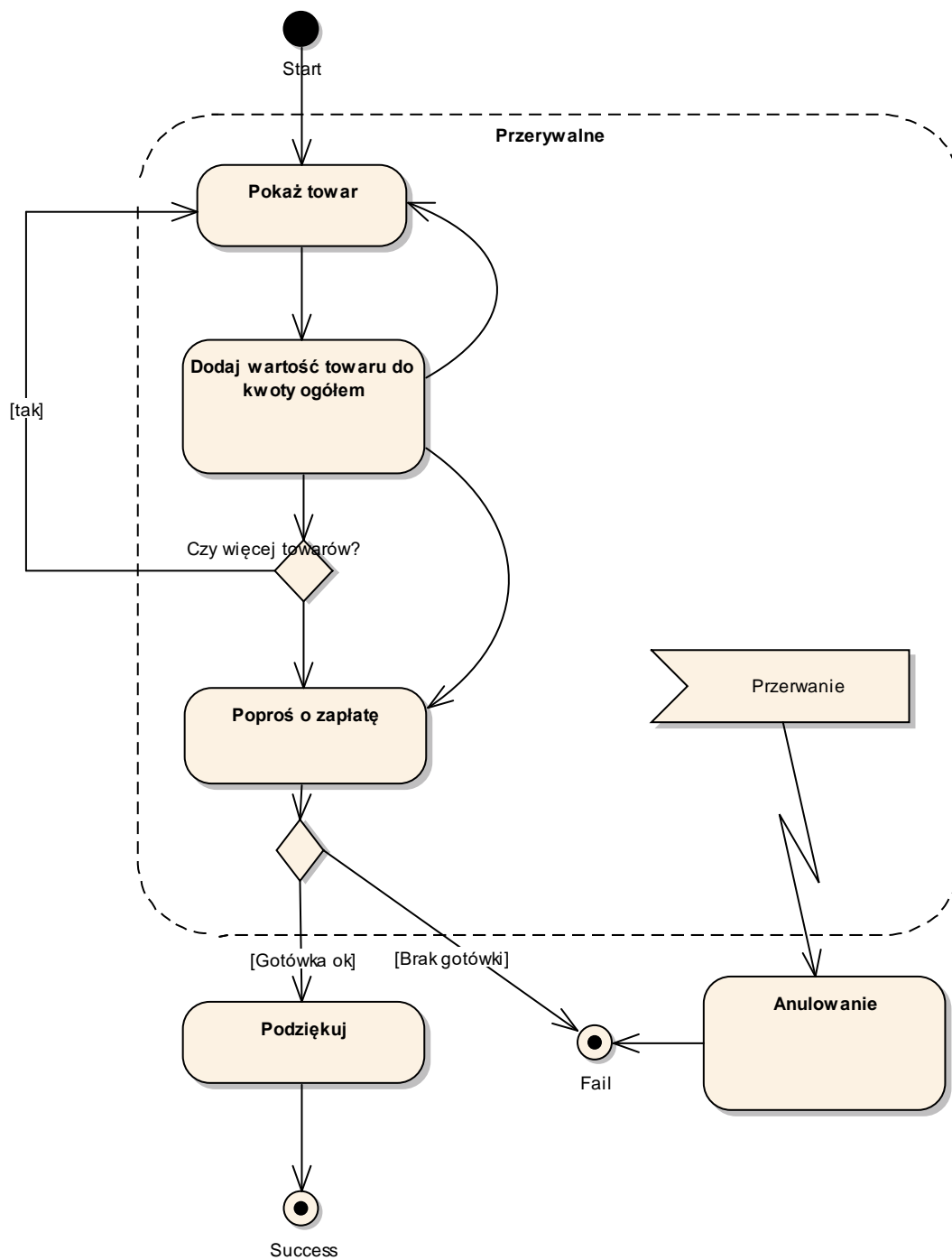
**Diagramy czynności (Activity Diagram)** służą do reprezentacji procesów, m.in.:

- przypadków użycia
- implementacji algorytmów

- Czynności vs akcje
  - Czynności – długotrwałe, podzielne, ogólne
  - Akcje – krótkotrwałe, niepodzielne, szczegółowe – nazwane czasownikowo (wprowadź/wybierz/zatwierdź/wydrukuj/aktualizuj/weryfikuj)
- Różnica w stosunku do diagramu stanów jeśli chodzi o semantykę bloków vs strzałek – tam bloczek = stan, strzałka = akcja; tu bloczek = akcja, strzałka – wyznacza następstwo akcji
  - Sygnały (zdarzenia) – wyślij, odbierz
  - Warian – „if”
  - Zdarzenia – send/receive
  - Regiony – na przykład „przerywalny”, pojawia się zdarzenie „przerwij”, anulowanie
  - Partycje – podział na aktorów

*Diagramy stanów i czynności wykorzystują niemalże ten sam formalizm do reprezentowania różnych kategorii diagramów.*

act Diagram czynności bez partycji



## 6 Diagramy sekwencji

**Diagramy sekwencji (Sequence Diagram)** służą do reprezentacji procesów, m.in.:

- Przypadków użycia
- Implementacji procesów

- Linie życia, paski aktywacji/ośrodki sterowania (execution specification)
- Typy obiektów
  - Boundary – widok
  - Control – kontroler
  - Entity – model
- Związek między diagramem sekwencji a diagramem klas – ustalanie typu obiektu
- Komunikat – wartość zwrótna  
wartość = komunikat( p1:P1, p2:P2, ... ) : typ
- lub przerywana strzałka zwrótna (EA – niekoniecznie)
- Singleton – jedynka w rogu, metoda statyczna – stereotyp „class”, „metaclass”
- Komunikat odnaleziony – „od nikogo”
- Create/destroy
- Ramki, można zagnieżdżać
  - Loop – pętla
  - Alt – if-then-else
  - Opt – if
  - Neg – czynność nieprawidłowa, wyjątek
  - Par - współbieżność
  - Ref – odwołanie do innej, nazwanej ramki
  - Sd – nazwana ramka

Przykładowy pseudokod:

```
public class Actor {
    public void XXXX() {
        while ( n < 10 ) {
            a.fooA();
        }
    }
}

public class A {
    public void fooA() {
        b.fooB();
        c.fooC();
    }
}

public class B {
    public void fooB() {
        d.fooD();
    }
}

public class C {
    public void fooC() {
```

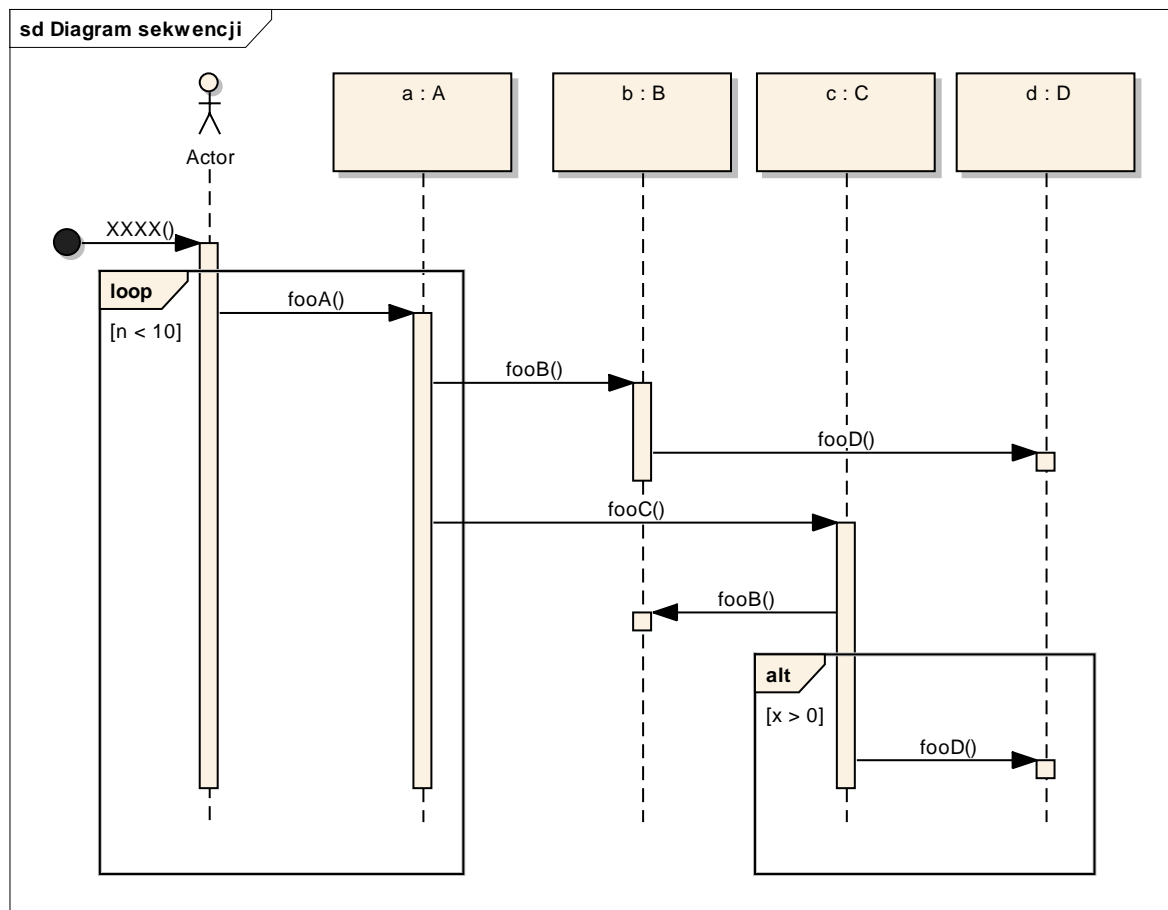


```

        b.fooB();
        if ( x > 0 )
            d.fooD();
    }
}

```

i jego diagram



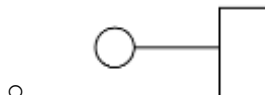
## Diagram czynności a diagram sekwencji

Diagram czynności	Diagram sekwencji
Dokumentacja procesów	Dokumentacja procesów
Może być używany do dokumentacji wysokopoziomowej (np. przypadki użycia) albo szczegółowej (implementacja)	Może być używany do dokumentacji wysokopoziomowej (np. przypadki użycia) albo szczegółowej (implementacja)
Oś czasu diagramu wyznaczona jest przez przejścia między akcjami, w szczególności łatwiej jest długi skomplikowany proces rozplanować na diagramie	Diagram ma jednoznaczną „oś czasu” – należy go czytać z góry na dół, długie skomplikowane procesy mogą być nieczytelne
W interpretacji implementacyjnej – jeśli diagram obejmuje więcej niż jedną metodę dwóch różnych obiektów to może być trudno określić które akcje wchodzą w skład której metody	W interpretacji implementacyjnej – ciągły blok na linii życia obiektu wyznacza granicę metody (funkcji)

## 7 Diagramy komponentów

**Diagramy komponentów (Component Diagram)** służą do reprezentacji komponentów systemów i ich powiązań

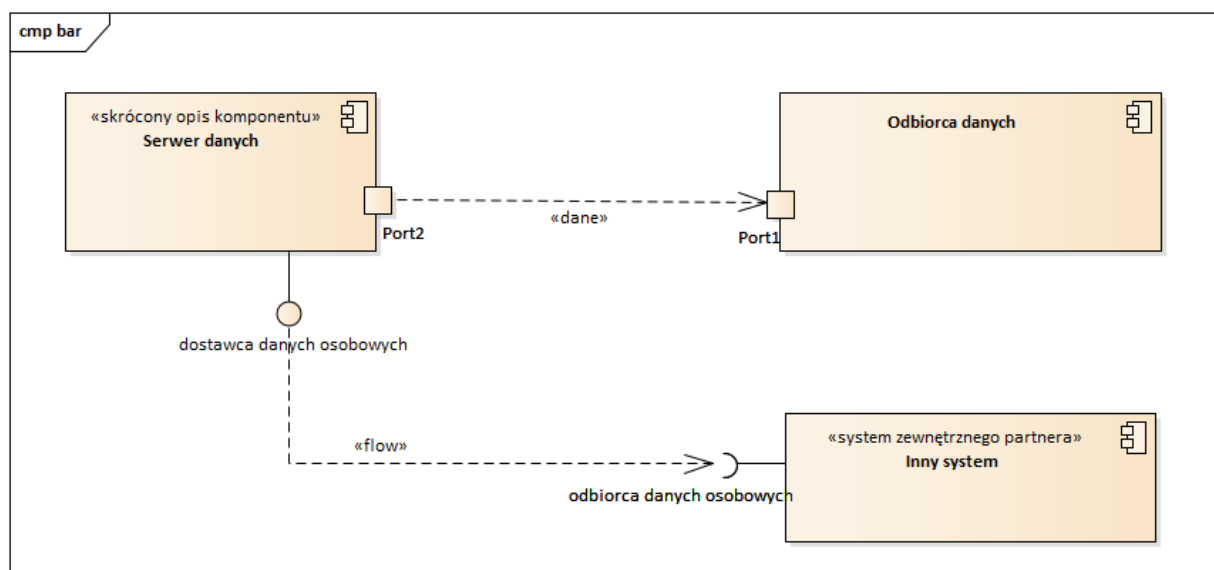
- przez komponent rozumie się tu coś więcej niż klasa. Może to być pakiet (zbiór klas), może być cały moduł systemu
- komponenty łączą się z innymi komponentami za pomocą portów



- dostawca usługi – port na komponencie który posiada implementację usługi dla innego komponentu



- odbiorca usługi – port na komponencie który korzysta z usługi dostarczanej przez inny komponent (na przykład zależy od danych produkowanych przez inny komponent)



## 8 Literatura

Wrycza, Marcinkowski, Wyrzykowski - Język UML 2.0 w modelowaniu systemów informatycznych



The Unified Modelling Language - <https://www.uml-diagrams.org/>