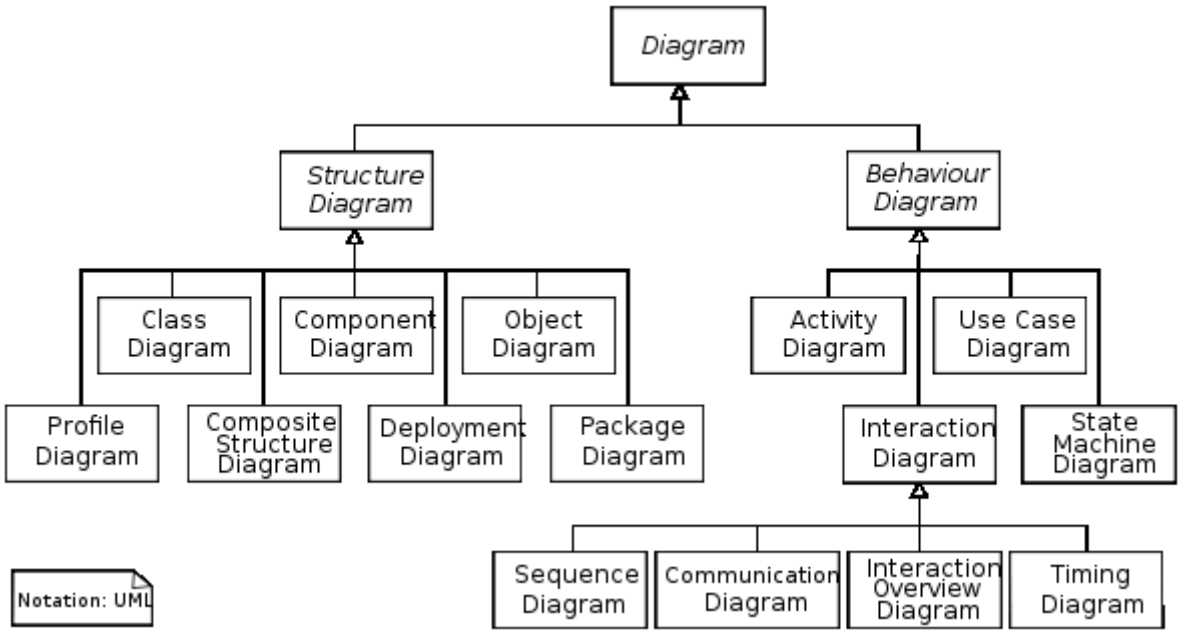
# Projektowanie obiektowe oprogramowania
# Wykład 2 - UML
# Wiktor Zychla 2017

## Spis treści

# 1 Wprowadzenie

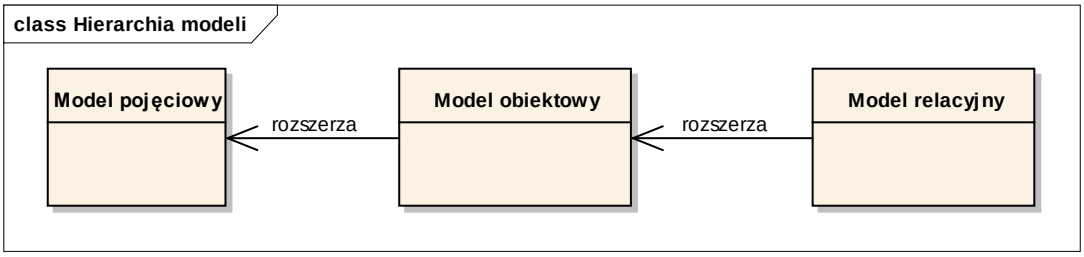



Dwie rodziny diagramów - diagramy **struktury** i diagramy **zachowań** (dynamiki)

# 2 Diagramy klas

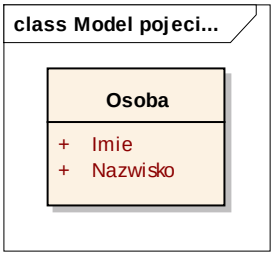
## 2.1 Hierarchia modeli

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.

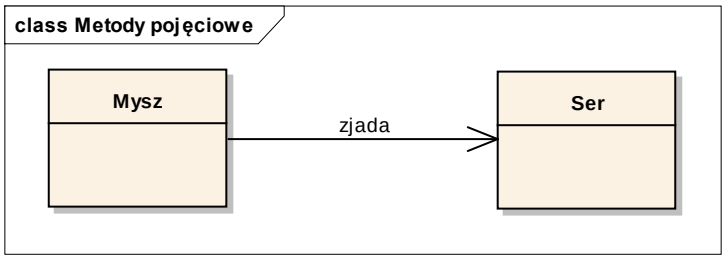



### 2.1.1 Diagram modelu pojęciowego

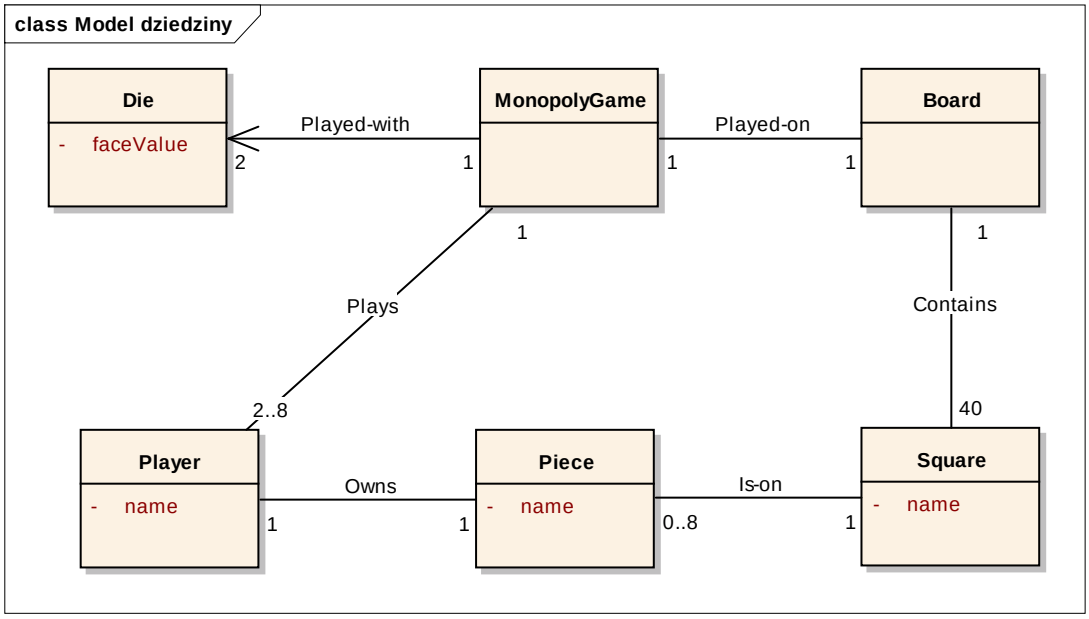
- Jest elementem projektu analitycznego
- Służy ustaleniu wspólnego języka w projekcie
- Służy weryfikacji podstawowych elementów struktury dziedziny projektu
- Pojęcia i atrybuty (tylko publiczne)




- 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



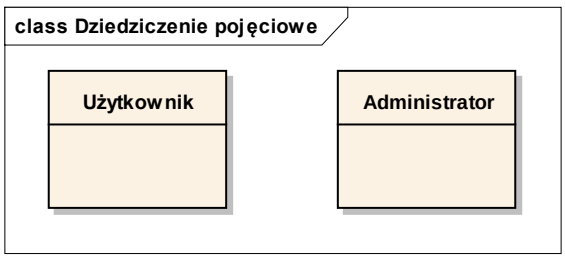

- Brak dziedziczenia i innych ograniczeń specyficznych dla struktury stricte obiektowej
- Brak metod (!)



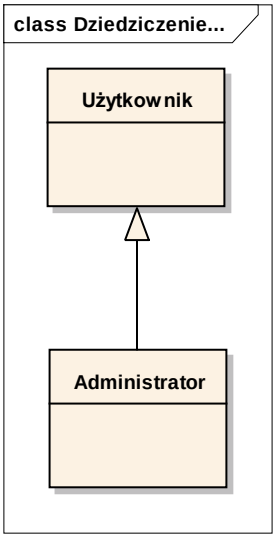

### 2.1.2 Diagram modelu obiektowego (diagram klas)

- 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:
    - **Usuwaniu** zbędnych pojęć, które są reprezentowane przez jedną i tę samą klasę (na przykład pojęcia Użytkownik i Administrator staną się jedną i tą samą klasą)
    - **Dodawaniu** 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)

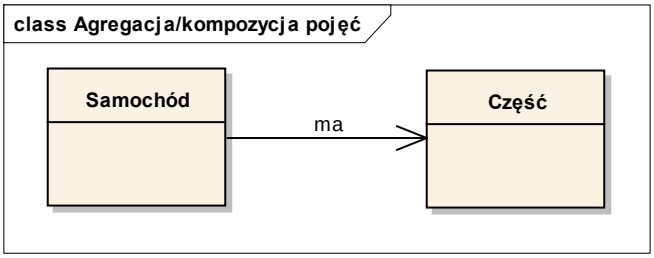
Przykład 1:



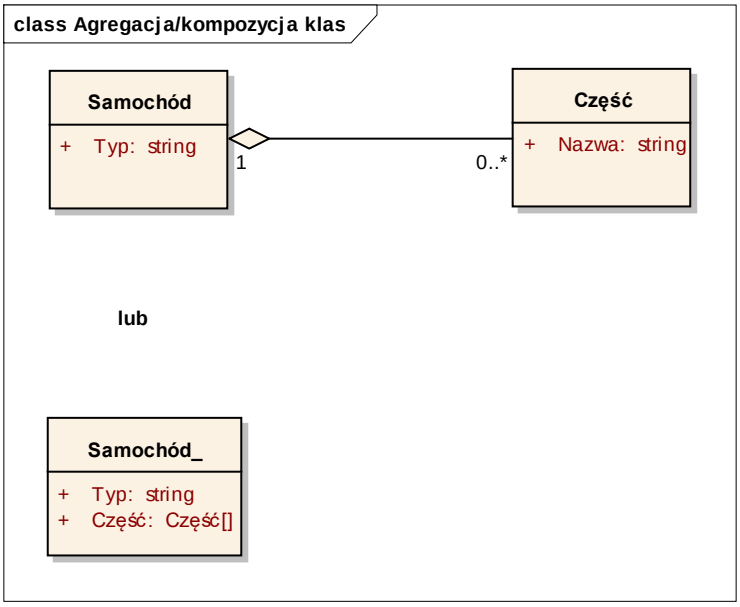
VS



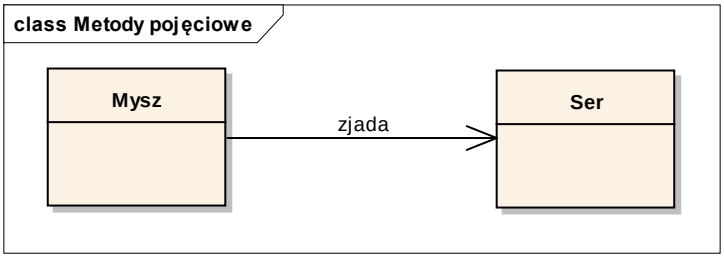

Przykład 2:



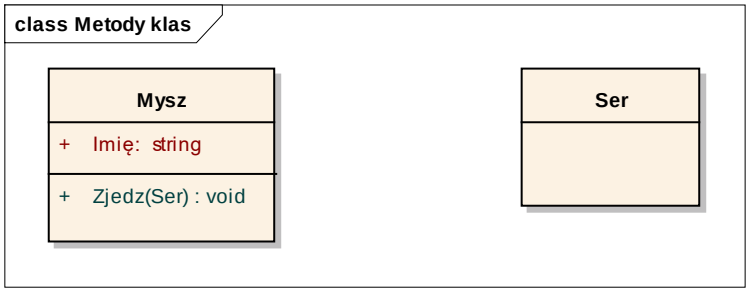

VS




Przykład 3:

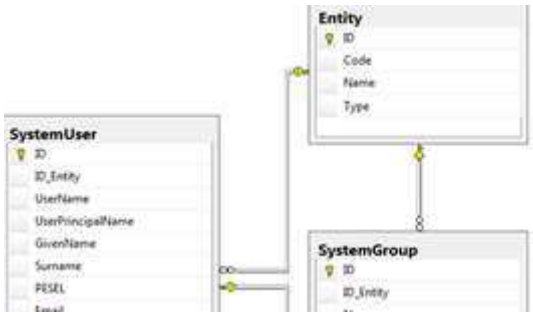



VS




### 2.1.3 Diagram modelu implementacyjnego (relacyjnego)

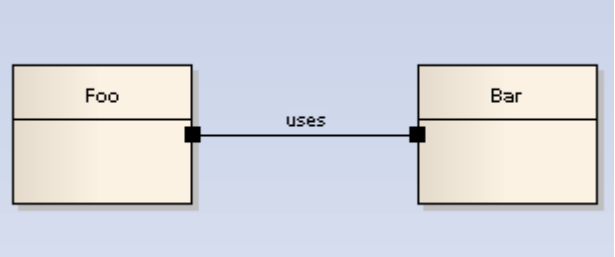
- To diagram reprezentujący strukturę relacyjnej bazy danych
- Reprezentuje fizyczną strukturę właściwą do utrwalania obiektów
- Punktem wyjścia jest diagram klas
- Podczas refaktoryzacji usuwa się z diagramu klas wszystkie te relacje, których nie da się reprezentować w świecie relacyjnym
    - Nie ma metod
    - Nie ma dziedziczenia, zamiast tego wybiera się jeden ze sposobów implementacji dziedziczenia
        - Table-per-type
        - Table-per-concrete-type
        - Table-per-hierarchy
    - Wprowadza się sztuczne identyfikatory główne (ID)
    - Nie ma relacji wiele-wiele, zamiast tego są pomocnicze tabele do reprezentacji takich relacji



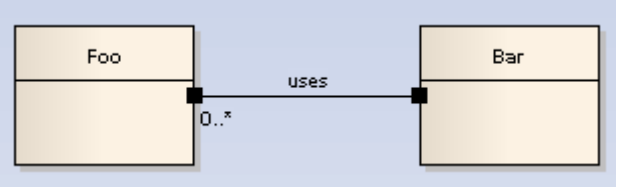

## 2.2 Jeszcze o formalizmie diagramów klas - klasy i asocjacje

- Zależności (strzałka przerywana) – brak informacji o rodzaju zależności, może być:
    - Tworzy
    - Wykorzystuje (zmienna lokalna)

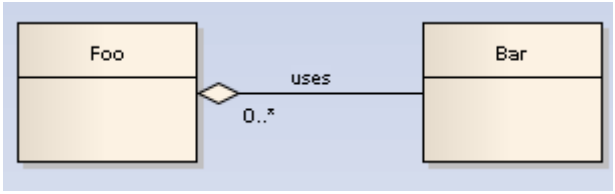
    - Wykorzystuje (parametr metody)
    - Nadklasa lub interfejs
- Nazwy asocjacji



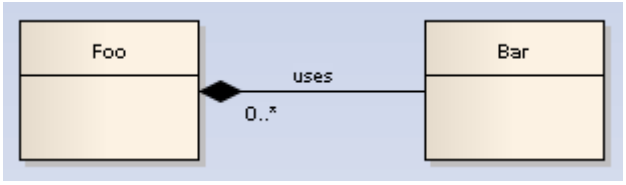

- Liczebność : 1, 1..*, 0..1, *, 0..*, n, 1..n, 0..n, n..m, n..*




- Agregacja vs kompozycja
    - Agregacja – luźniejsza

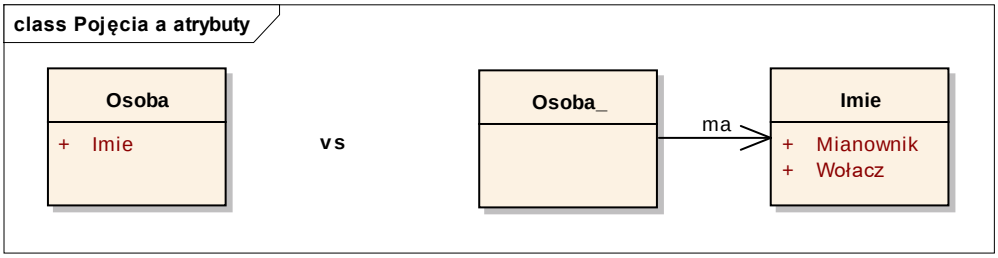



    - Kompozycja - ściślejsza
        - Instancja reprezentująca część może należeć tylko do jednej instancji złożonej
        - Czas życia części jest powiązany z czasem życia całości

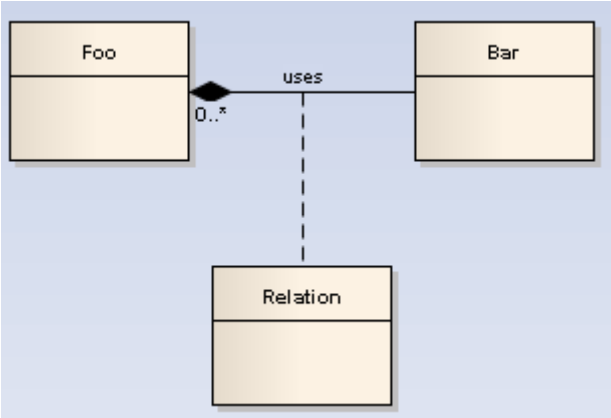



## 2.3 Składowe

- 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

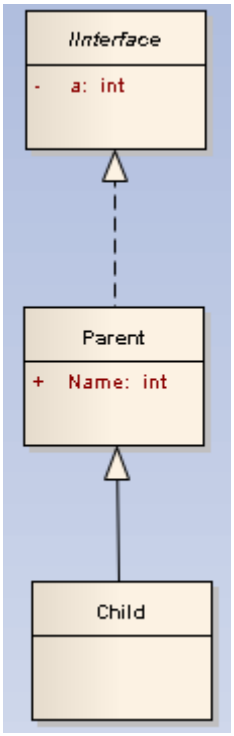



- Klasa asocjacyjna – do modelowania relacji wiele-wiele



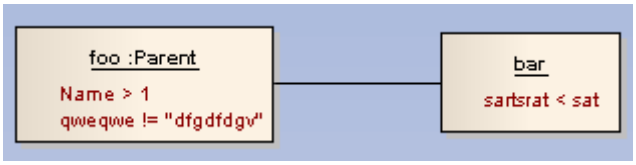

## 2.4 Dziedziczenie

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



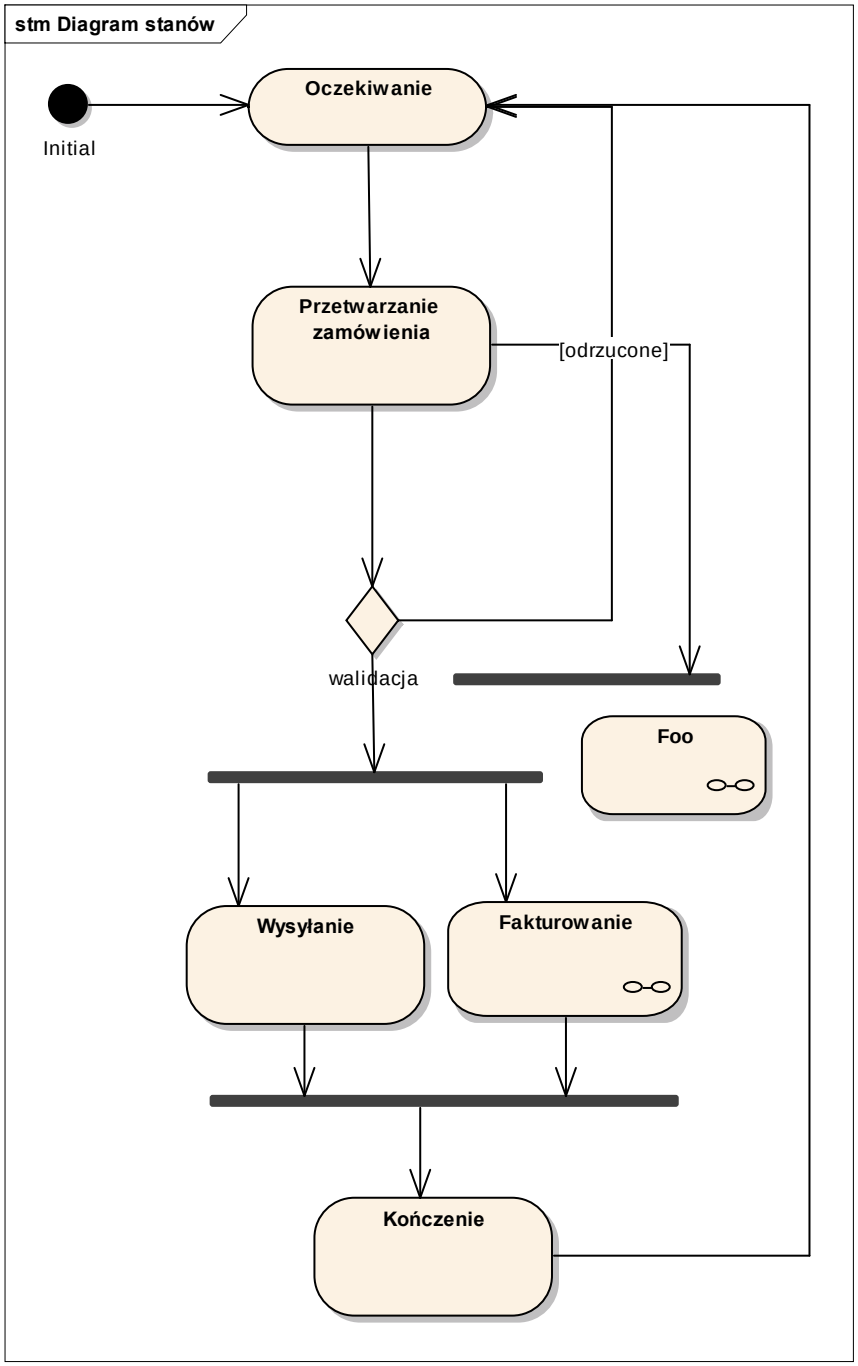
# 3 Diagramy obiektów

- Migawka systemu
- EA: Advanced / Instance Classifier – umożliwia wybór typu dla instancji
- EA: Advanced / Set Run State – umożliwia określenie stanu obiektu




# 4 Diagramy stanów

- 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
    - Stany – oczekiwanie, przetwarzanie
    - Wariant – nazwany
    - Zrównoleglanie – wysyłanie, fakturowanie
    - Stan kompozytowy

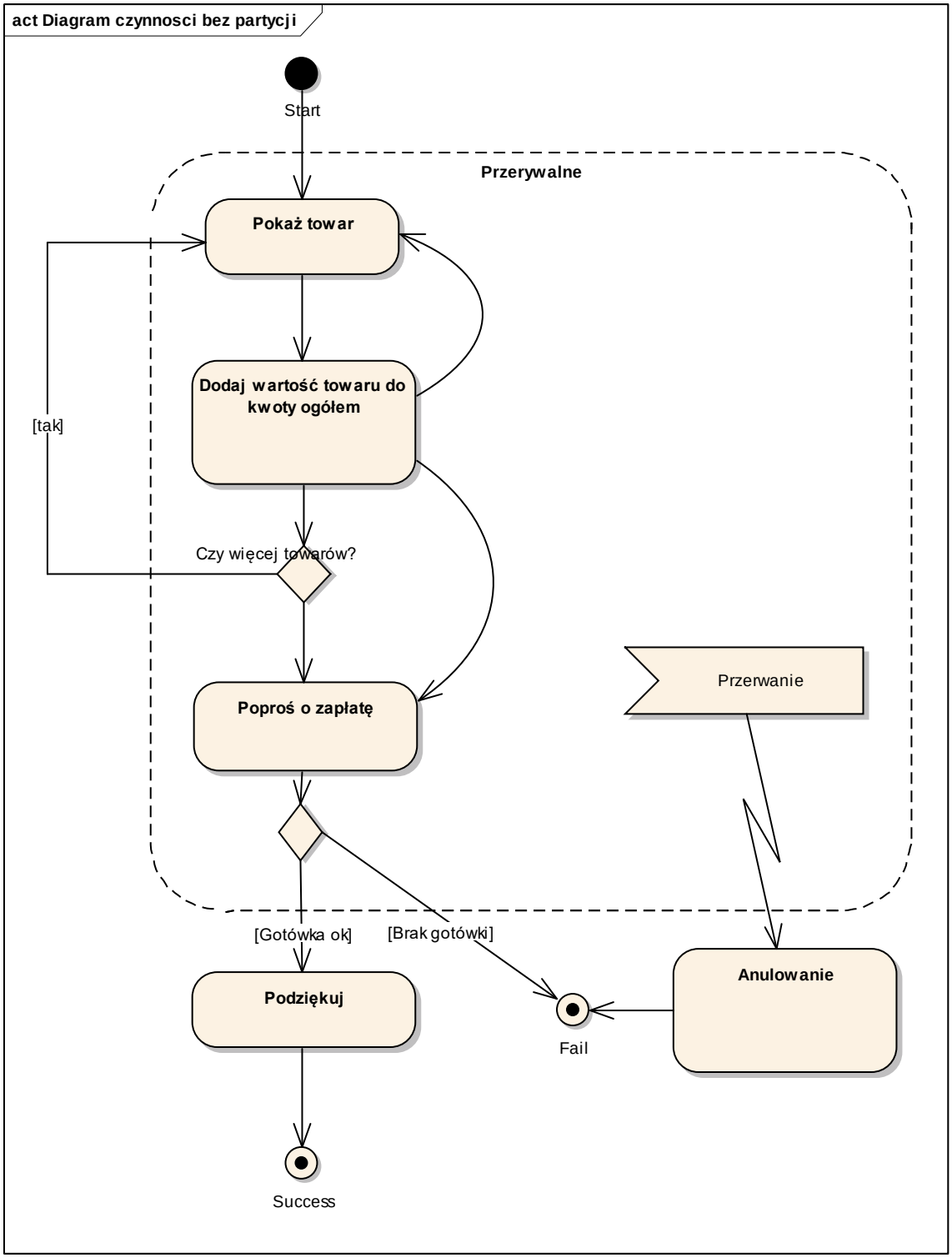



# 5 Diagramy czynności

- 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
  - Wariant – „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.*



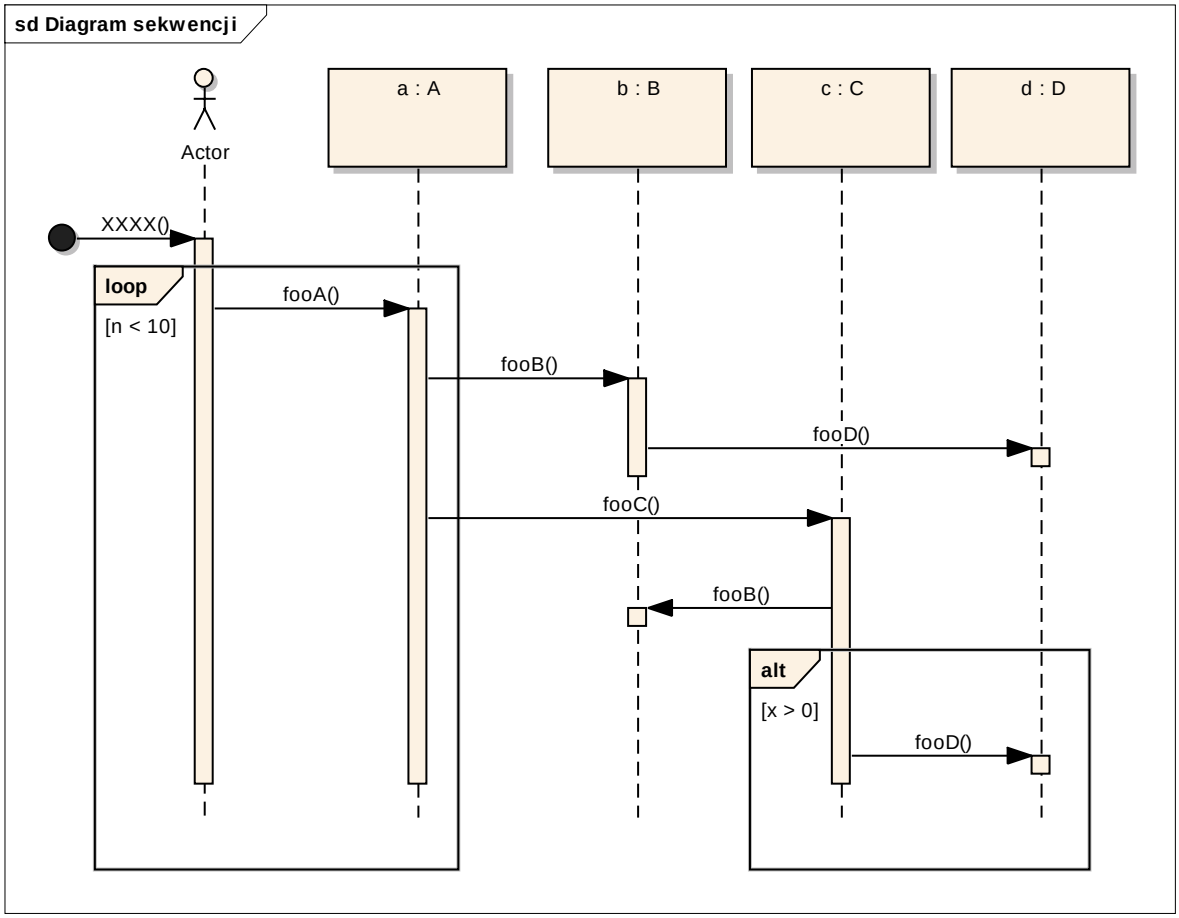
# 6 Diagramy sekwencji

- 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 zwrotna
  wartość = komunikat( p1:P1, p2:P2, … ) : typ
- lub przerywana strzałka zwrotna (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




# 7 Literatura

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