

# **Modelowanie i analiza obiektowa**

Wykład 5

Analiza przypadków użycia

Wykład 6

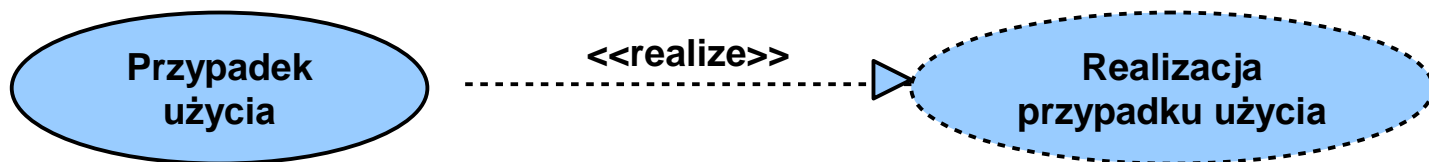
Projektowanie klas

# Plan wykładu

- **Realizacja przypadku użycia**
- **Realizacja przypadku użycia w fazie analizy**
  - **Klasy analizy**
  - **Analiza zachowania na diagramach interakcji**
  - **Faza analizy a wymagania нефunkcjonalne**

# Realizacja przypadku użycia

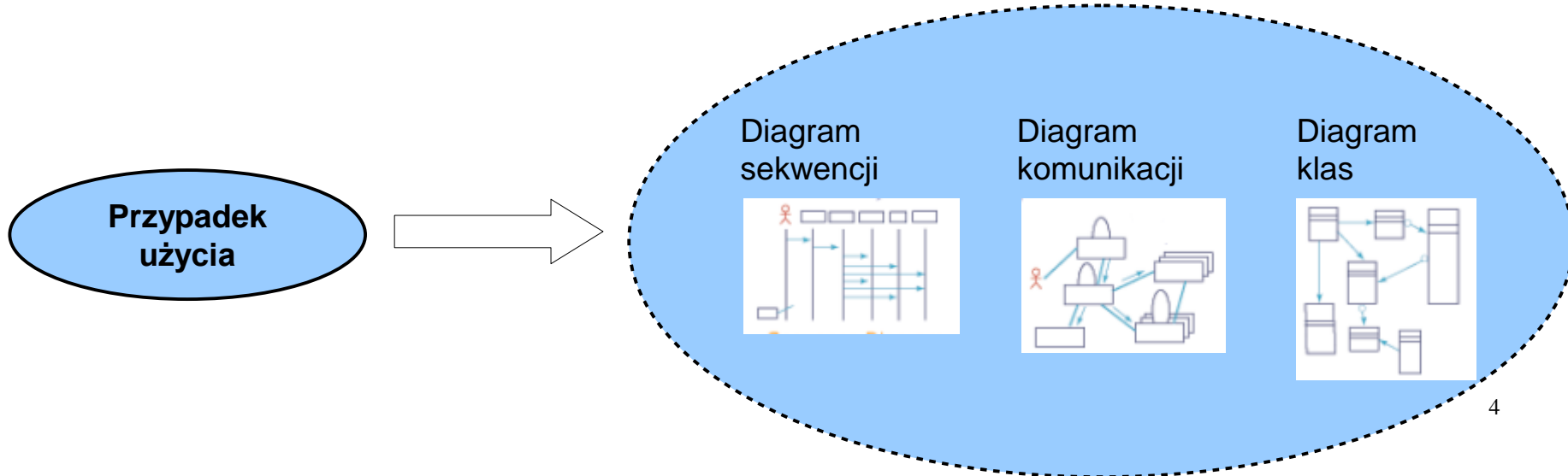
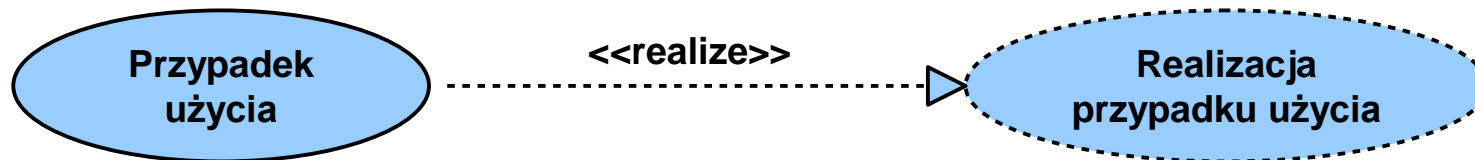
- Opis sposobu, w jaki konkretny przypadek użycia jest realizowany w modelu projektowym.
- Określa powiązanie pomiędzy modelem przypadków użycia a modelem projektowym.
- Określa, jakie klasy muszą zostać zbudowane, aby zaimplementować dany przypadek użycia.



# Elementy realizacji przypadku użycia

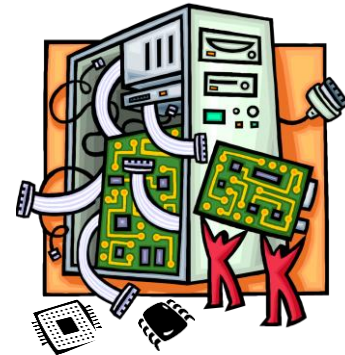
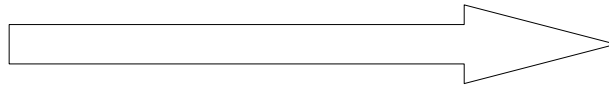
Model przypadków użycia

Model projektowy



# Analiza przypadku użycia

- Jest to pierwszy etap realizacji przypadku użycia
- Określa, jak zewnętrzne zachowanie systemu przekłada się na współdziałanie jego elementów logicznych.
- Elementami logicznymi są klasy analizy



# Kroki analizy przypadków użycia

1: Dla każdej realizacji Przypadku Użycia

1.1: Znalezienie klas na podstawie analizy zachowania

1.2: Przydzielenie klasom odpowiedzialności

2: Określanie własności klas analizy

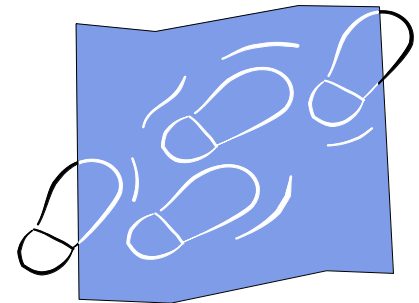
2.1: Opisanie odpowiedzialności

2.2: Opisanie atrybutów i asocjacji

2.3: Przypisanie mechanizmu analizy

Krok 3: Unifikacja klas

Krok 4: Kontrola



# Krok 1.1: Wyszukiwanie klas (analizy) dla każdej realizacji przypadku użycia

## Gdzie szukać kandydatów na klasy?

Specyfikacja przypadku użycia

Słownik pojęć

Dokument wymagań klientów



Twórczy proces, który w pewnych sytuacjach powinien odbywać się w obecności ekspertów z dziedziny problemowej

Klasy powinny odzwierciedlać modelowaną dziedzinę biznesową oraz słownik systemu używany do projektowania

# Nazwa klasy

- Powinna pochodzić z dziedziny problemowej i odzwierciedlać to co reprezentuje (Student, Wykładowca, Kurs)
- Nazwa powinna być jednoznaczna i unikatowa (rzeczownik – osoba, miejsce, rzecz)
- Potencjalne nazwy klas znajdują się w dokumentach wymagań



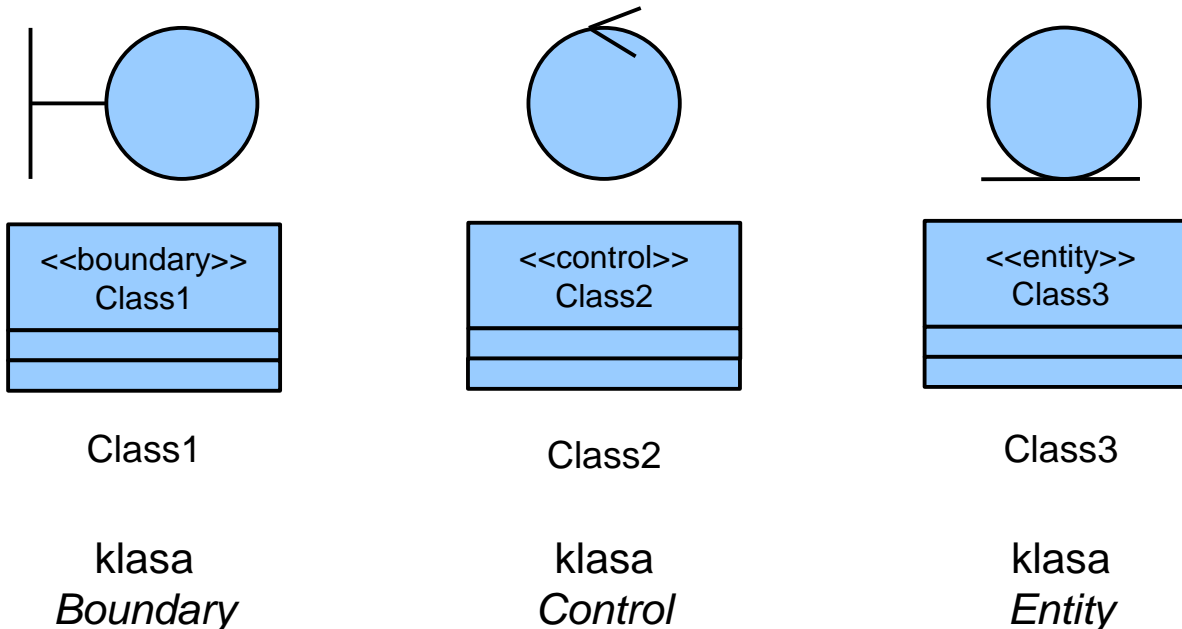


# Czym są klasy analizy

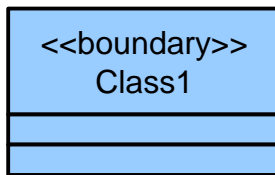
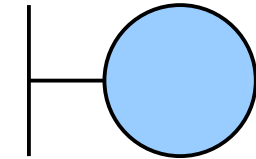
- Klasy analizy reprezentują koncepcyjny model elementów systemu, które posiadają pewne zachowanie i zadania. Jest to pierwszy szkic modelu systemu.
- Klasy analizy reprezentują trzy aspekty systemu, które często podlegają zmianom:
  - Granice pomiędzy systemem i aktorami,
  - Dane, których system używa.
  - Sposób sprawowania kontroli na systemem (przetwarzaniem)
- Klasy analizy są swego rodzaju prototypami klas, część z nich stanie się podsystemami, komponentami, klasami, inne zostaną odrzucone w dalszym procesie projektowym.

# Klasy analizy

Modelowanie zachowania przypadku użycia w fazie analizy podzielone jest pomiędzy trzy typy klas:



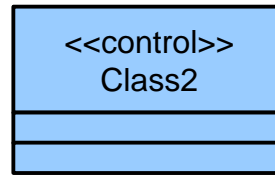
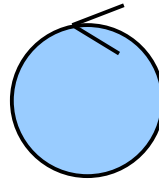
# Klasy analizy



Class1

klasa  
*Boundary*

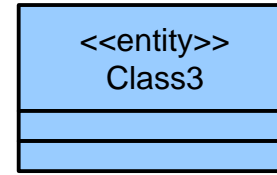
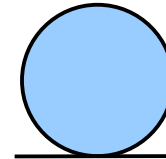
stanowi granicę  
pomiędzy systemem  
a jego aktorami



Class2

klasa  
*Control*

obrazuje element  
kontrolny, sterujący  
działaniem



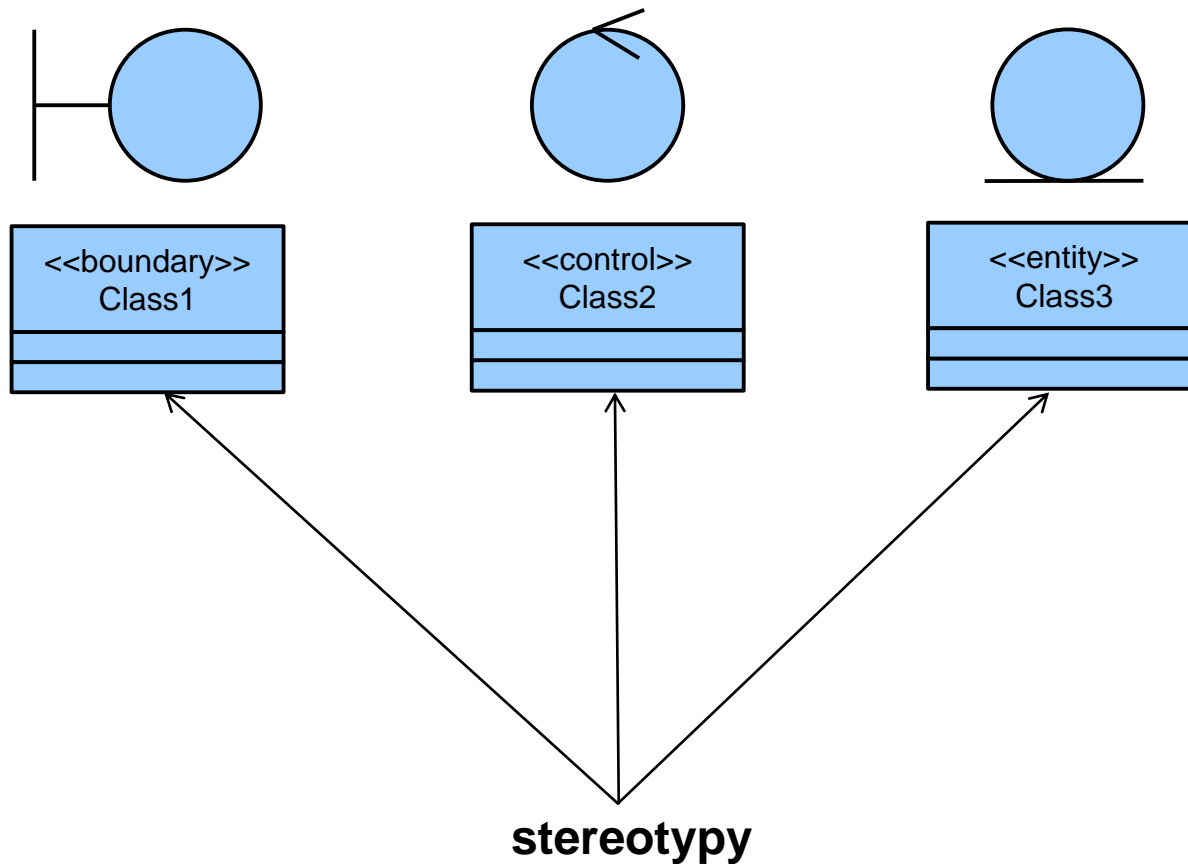
Class3

klasa  
*Entity*

informacja, którą  
wykorzystuje  
system

# Klasy analizy

## Reprezentacje graficzne



# Przypadek użycia a klasy analizy

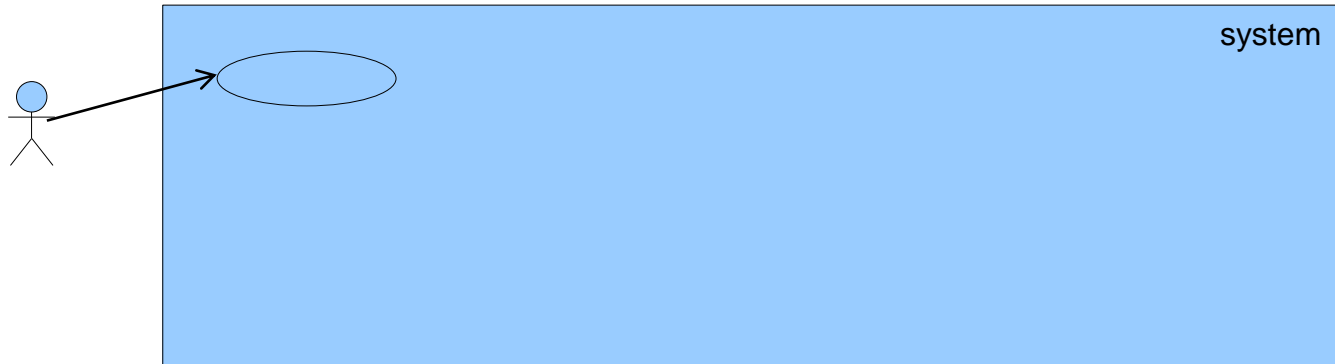


Diagram przypadków użycia

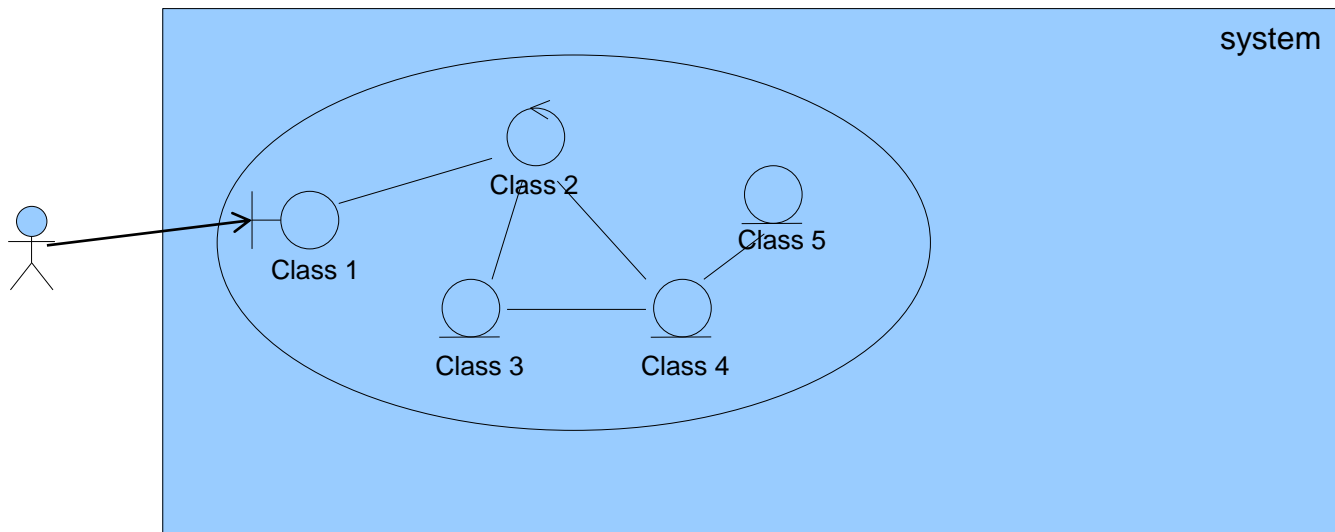
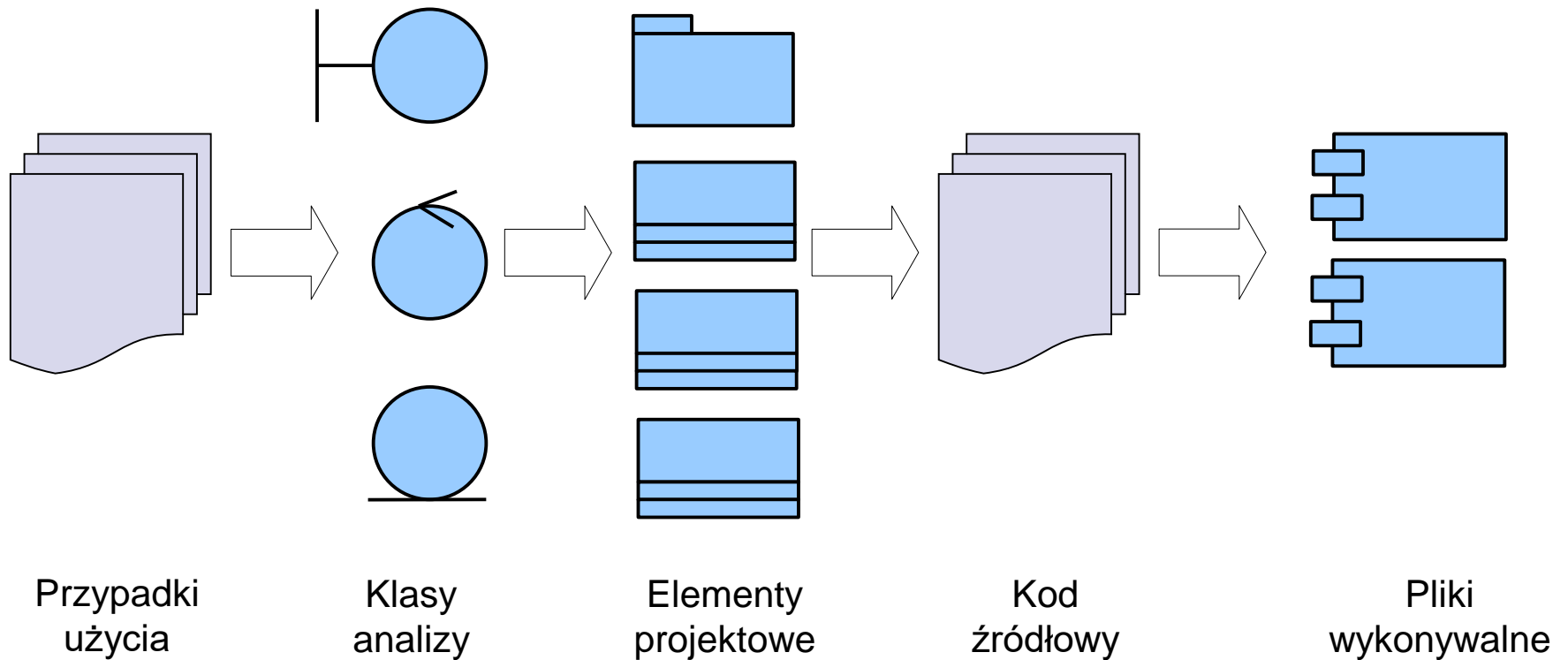
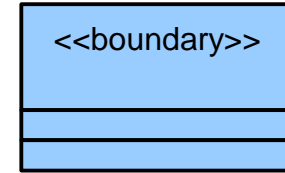
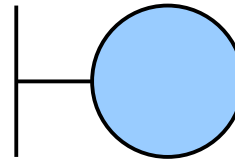


Diagram realizacji przypadku użycia przy pomocy klas analizy

# Czym są klasy analizy



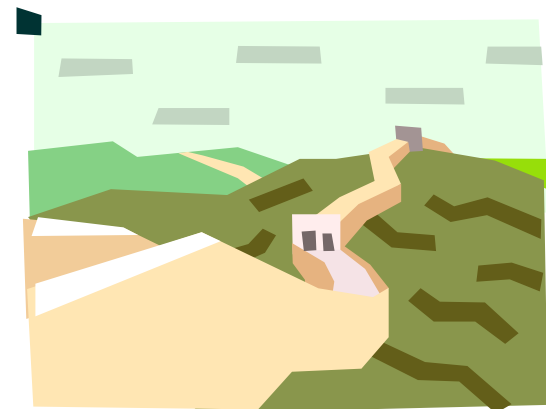
# Klasa **Boundary**



- Różne typy (Zależne od środowiska)
  - Klasy interfejsu użytkownika
  - Klasy interfejsu z innym systemem
  - Klasy interfejsu sprzętowego
- Założenie
  - Jedna klasa **Boundary** dla pary:  
Aktor – Przypadek użycia.

# Klasa **Boundary** – cechy

- Modeluje interakcje pomiędzy otoczeniem a systemem.
- Modeluje te części systemu, które są zależne od otoczenia
- Izolują wnętrze systemu od wpływu środowiska (i vice versa)



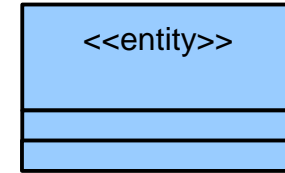
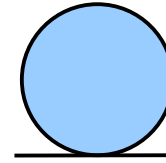


# Odkrywanie klas Boundary - wskazówki

- Klasy interfejsu użytkownika
- Skup się na tym jak informacja jest prezentowana użytkownikowi (nie zwracaj uwagi na szczegóły)
- Klasy interfejsu do innego systemu lub interfejsu urządzenia
- Jaki protokół komunikacji musi być zdefiniowany (nie koncentruj się na tym w jaki sposób go zaimplementować)



# Klasa Entity



- Logiczna struktura danych.
- Reprezentuje i zarządza informacjami przechowywanymi przez system.
- Zazwyczaj trwała
- Niezależna od środowiska zewnętrznego systemu (aktora)
- Zazwyczaj nie odpowiada tylko jednej realizacji przypadku użycia.



# Klasa Entity

Źródła, w których należy wyszukiwać klasy Entity:

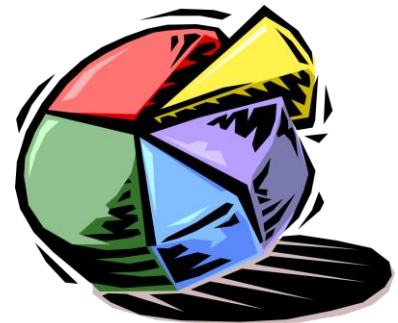
- Słownik
- Model dziedziny biznesowej
- Przepływ zdarzeń przypadku użycia
- Podstawowe abstrakcje



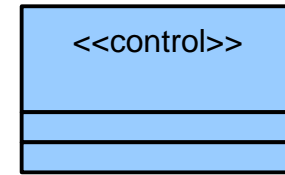
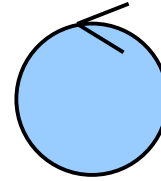
# Klasa Entity

Wyszukiwanie klasy Entity:

- Wyszukanie rzeczowników (fraz rzeczownikowych) w przepływie zdarzeń (specyfikacja przypadku użycia).
- Utworzenie listy kandydatów
- Filtrowanie listy (znalezienie rzeczowników oznaczających to samo, niejednoznacznych, zbyt ogólnych, usunięcie aktorów, atrybutów, operacji)



# Klasa Control



- Koordynator przypadku użycia
- Definiuje logikę działania (kolejność przesyłania komunikatów) mechanizm obsługi błędów czy transakcji dla przypadku użycia
- Rozdziela klasy Boundary i Entity, dzięki czemu łatwiej będzie w przyszłości wprowadzić zmiany
- Jedna klasa Control na Przypadek Użycia
- Zależna od Przypadku Użycia, niezależna od środowiska zewnętrznego
- Zazwyczaj ma kilka stanów zależnych od miejsca w przepływie zdarzeń.



# Kroki analizy przypadków użycia

1: Dla każdej realizacji Przypadku Użycia

1.1: Znalezienie klas na podst. analizy zachowania

1.2: Przydzielenie klasom odpowiedzialności

2: Określanie własności klas analizy

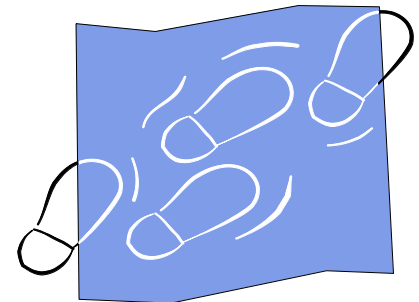
2.1: Opisanie odpowiedzialności

2.2: Opisanie atrybutów i asocjacji

2.3: Przypisanie mechanizmu analizy

Krok 3: Unifikacja klas

Krok 4: Kontrola



# Przydzielenie klasom odpowiedzialności - wskazówki

- Klasy Boundary
  - zachowanie, z którym związana jest komunikacja z aktorem
- Klasy Entity
  - zachowanie związane z danymi, przechowywanymi w systemie
- Klasy Control
  - Zachowanie specyficzne dla Przypadku Użycia



**Nie należy modelować interakcji pomiędzy aktorami, są oni poza granicami systemu.**

# Diagramy interakcji

**Diagramy interakcji**, stanowiące jeden z rodzajów diagramów dynamicznych, pozwalają na utworzenie opisu interakcji obiektów systemu podczas realizacji danego zadania: Przypadku Użycia czy też jednego konkretnego ciągu zdarzeń dla danego Przypadku Użycia

UML posiada dwa rodzaje diagramów interakcji

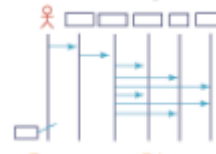
diagramy komunikacji (*communication diagrams*, w UML 1.x *collaboration diagrams*)

Diagram komunikacji



diagramy sekwencji (ang. *sequence diagrams*)

Diagram sekwencji





# Diagramy interakcji

- Konstruowanie diagramów interakcji nie musi być stosowane dla wszystkich Przypadków Użycia. Mogą się one okazać szczególnie użyteczne do:
  - komunikacji wewnątrz zespołu projektowego (jak zresztą wszystkie rodzaje diagramów),
  - rozważenia opcjonalnych realizacji w “trudnych przypadkach”.
- Ponadto, niektóre narzędzia CASE potrafią wykorzystać te diagramy do generacji kodu, co może stanowić ważny powód dla ich konstruowania.

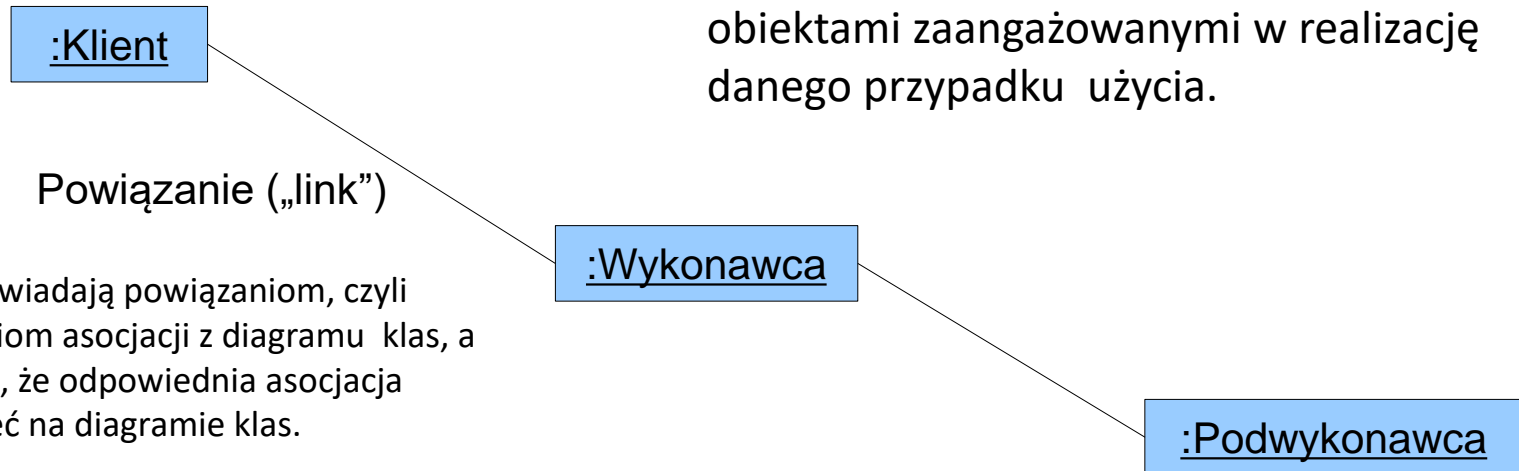
# Diagramy interakcji

- Oba rodzaje diagramów, bazując na danym diagramie klas, pokazują prawie tę samą informację, ale w nieco inny sposób.
- Decyzja, który rodzaj diagramów konstruować, zależy od pożądanego aspektu interakcji.

# Diagram komunikacji\*

Diagramy komunikacji pokazują w jaki sposób system realizuje dany przypadek użycia i jakie elementy (**obiekty**) są zaangażowane w tę realizację.

Diagramy komunikacji mogą dodatkowo pokazywać interakcje zachodzące między obiektami zaangażowanymi w realizację danego przypadku użycia.



Powiązanie („link”)

Linki odpowiadają powiązaniom, czyli wystąpieniom asocjacji z diagramu klas, a to oznacza, że odpowiednia asocjacja musi istnieć na diagramie klas.

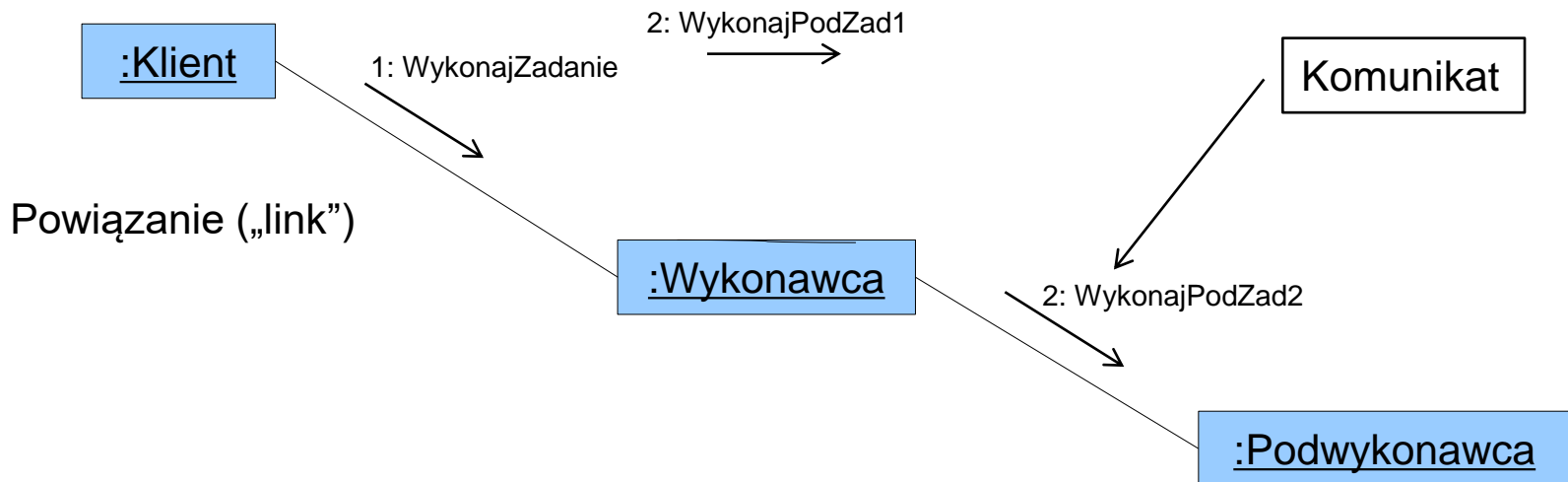
\*) w UML 1.x znany jako diagram współpracy (kolaboracji)

# Diagram komunikacji

- Prosty diagram komunikacji, bez uwidaczniania interakcji między obiektami, stanowi coś w rodzaju “wystąpienia fragmentu diagramu klas” – pokazuje aktora, relewantne obiekty i powiązania między nimi.
- Możliwe jest pokazanie więcej niż jednego obiektu danej klasy.

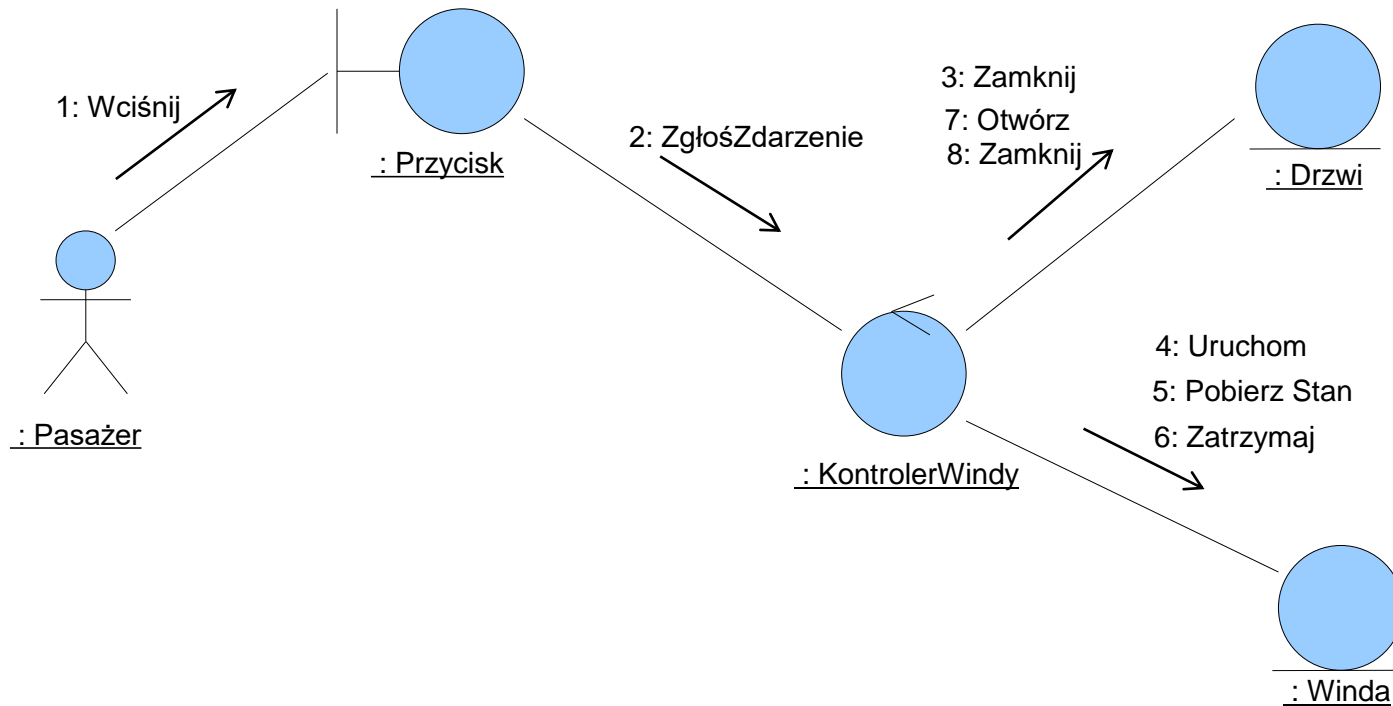
# Interakcja na diagramach komunikacji

Diagramy komunikacji mogą dodatkowo pokazywać interakcje zachodzące między obiektami zaangażowanymi w realizację danego **przypadku użycia**. Sekwencja interakcji oznacza tu sekwencję komunikatów przesyłanych między współpracującymi obiektami.



Komunikaty przedstawiane są tu w postaci etykiet strzałek rysowanych wzdłuż linków między współpracującymi obiektami.

# Diagram komunikacji - winda



# Rodzaje interakcji

## Sekwencyjna:

- Tylko jeden aktor może zainicjować sekwencję komunikatów i w danym momencie tylko jeden obiekt może “działać”.
- Obiekt rozpoczyna tzw. “aktywne życie” (*live activation*) w momencie otrzymania komunikatu.
- Może wysłać odpowiedź do nadawcy komunikatu.
- W międzyczasie może prowadzić obliczenia czy też wysyłać komunikaty do innych obiektów.
- Wysyłając komunikat do innego obiektu nadal pozostaje w aktywnym stanie, ale jego własna działalność zostaje zawieszona do czasu otrzymania odpowiedzi na wysłany komunikat – wysyłanie komunikatu związane jest tu z przekazywaniem sterowania do odbiorcy komunikatu.
- W każdym momencie istnieje w systemie stos aktywnych obiektów:
  - na szczycie stosu znajduje się ten obiekt, który aktualnie “działa”,
  - wysłanie odpowiedzi na komunikat powoduje zdjęcie obiektu ze stosu.

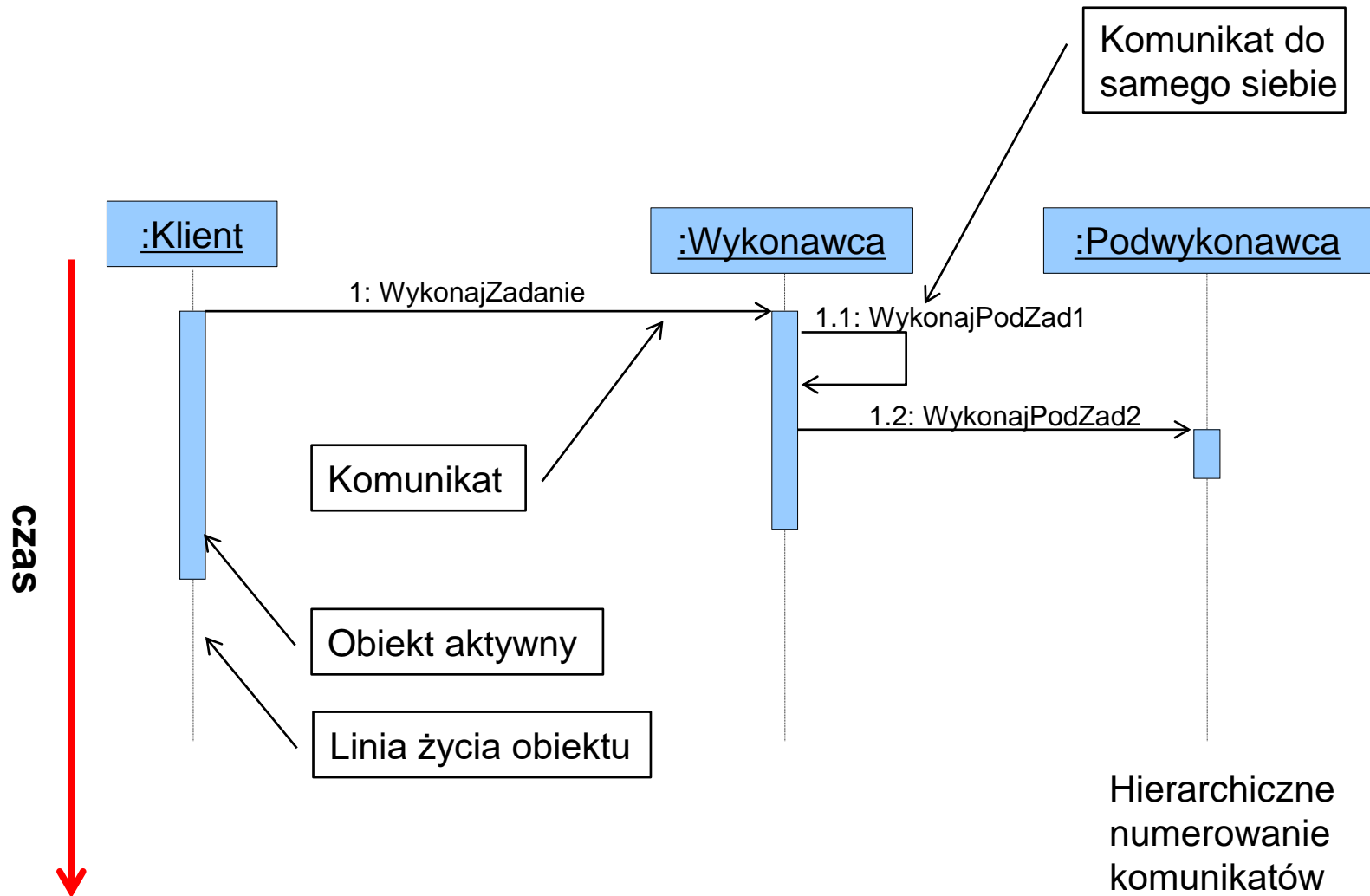
# Rodzaje interakcji

## Współbieżna:

- Wolna od ograniczeń interakcji sekwencyjnej.
- Pozwala na aktywne działanie wielu obiektów, które mogą wysyłać wiele komunikatów jednocześnie.
- Takie asynchroniczne działanie jest wykorzystywane w systemach wielowątkowych (np. rozproszonych).



# Diagramy sekwencji



# Diagramy sekwencji

Oznaczenia (notacja) interakcji:

Synchroniczna (*synchronous*),



Asynchroniczna (*asynchronous*),



Jednostronna (*flat*),



Powrót (*return*).



Nadawca zawiesza działanie oczekując na odpowiedź odbiorcy

Nadawca nie oczekuje na odpowiedź, pracuje dalej (w tym wysyła inne komunikaty)

Nadawca wysyła komunikat do odbiorcy i kończy działanie nie oczekując na odpowiedź

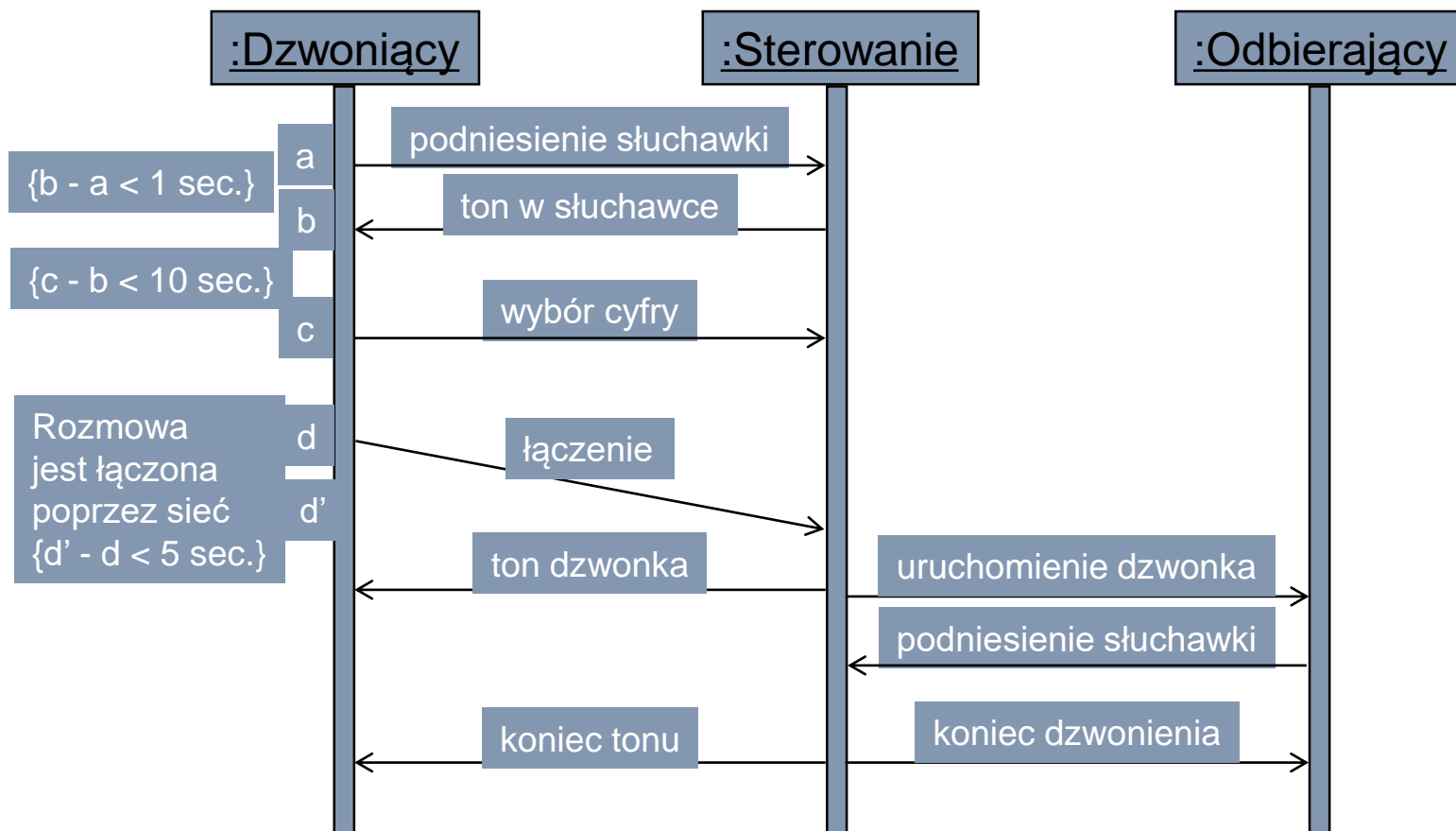
Zakończenie komunikatu i przekazanie sterowania do odbiorcy.  
**Powrót nie jest komunikatem.**

# Diagramy sekwencji

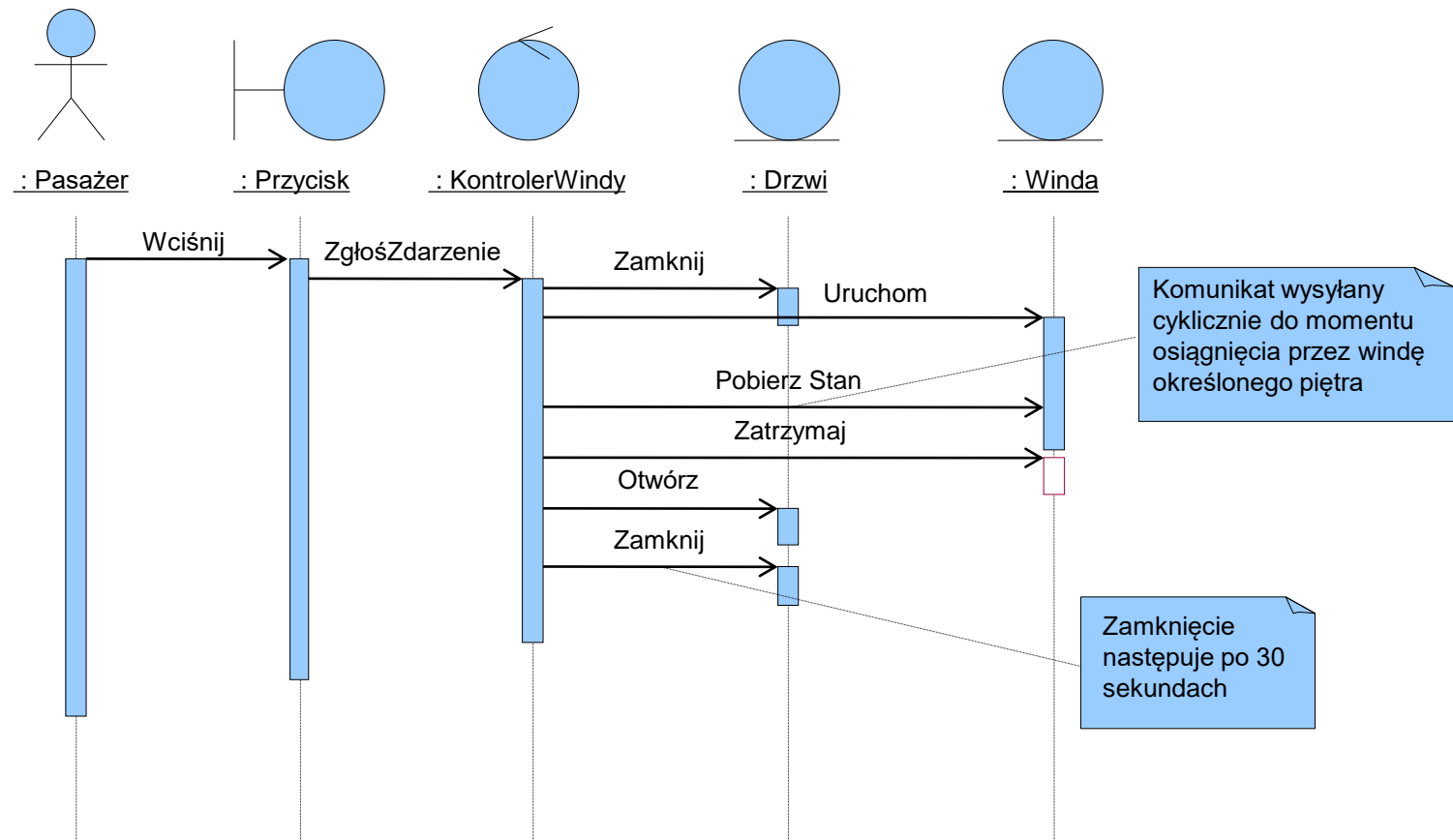
- Diagramy sekwencji nie pokazują linków między współpracującymi obiektami, ale można to wydedukować w oparciu o zaznaczone komunikaty.
- Diagramy sekwencji, wyraźniej niż diagramy komunikacji, pokazują przekazywanie sterowania.
- Istotnym elementem opisu jest czas, który chronologicznie segreguje komunikaty.

# Nakładanie ograniczeń na przepływ czasu

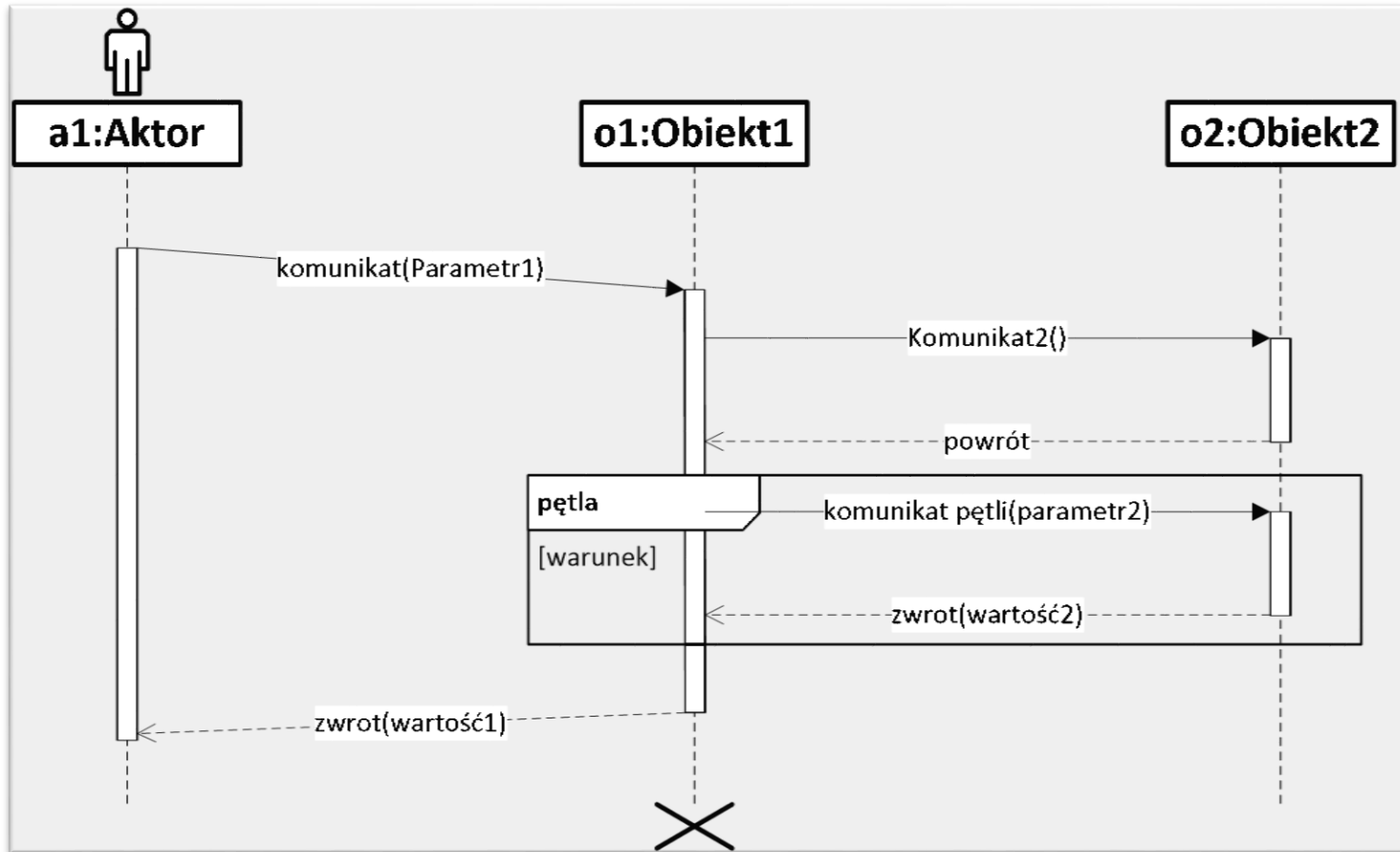
Główna przewaga diagramów sekwencji nad diagramami kolaboracji przejawia się w ich zdolności do graficznego prezentowania przepływu czasu, a nawet do podawania ograniczeń czasowych. Taka możliwość może mieć duże znaczenie dla opisu systemów czasu rzeczywistego.



# Diagram sekwencji - przykład



# Diagramy sekwencji w UML 2.0



# Diagramy sekwencji a diagramy komunikacji

**Diagramy są semantycznie równoważne, jednak pokazują informacje w różny sposób.**

## **Diagramy sekwencji:**

- Bezpośrednio pokazują kolejność komunikatów
- Uwypuklają przepływ sterowania
- Lepiej modelują złożone scenariusze oraz specyfikację dla systemów czasu rzeczywistego.

## **Diagramy komunikacji:**

- Kładą nacisk na powiązania
- Uwypuklają wzorzec współpracy (kto z kim)
- Uwypuklają efekty działania danego obiektu
- Łatwiejsze do narysowania w trakcie burzy mózgów (np. z wykorzystaniem kart CRC), której wynikiem jest opis działania systemu podczas realizacji przypadku użycia

# Krok 2: Określanie własności klas analizy

Krok 2.1: Opisanie odpowiedzialności

Krok 2.2: Opisanie atrybutów i związków

Krok 2.3: Przypisanie mechanizmu analizy



Dokumentowanie co klasa posiada i co robi.



# Opisanie odpowiedzialności klas

## Odpowiedzialność klasy

- działanie, które obiekt klasy jest zobowiązany wykonać.
- Informacje jakie obiekt posiada, którymi zarządza i które udostępnia

## Źródła

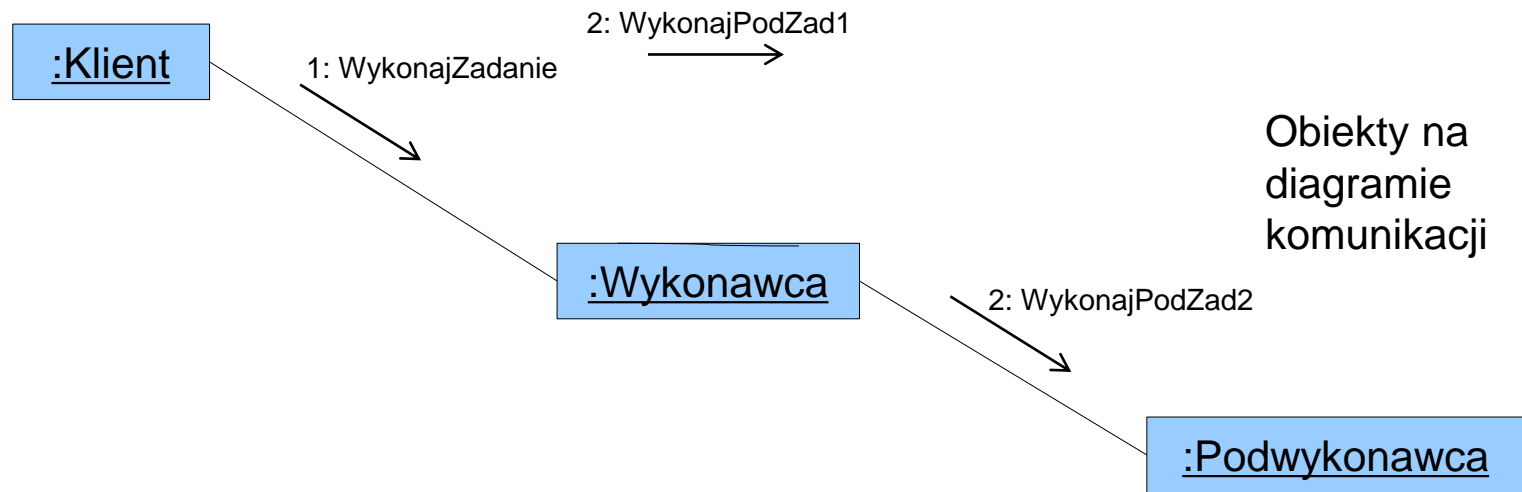
- Diagramy interakcji
- Wymagania нефункциональные

Należy pamiętać o tym, że w procesie projektowania wyniki analizy (klasy, ich odpowiedzialności) w wielu przypadkach ulegną zmianie.

# Odpowiedzialności klas a interakcja na diagramach interakcji

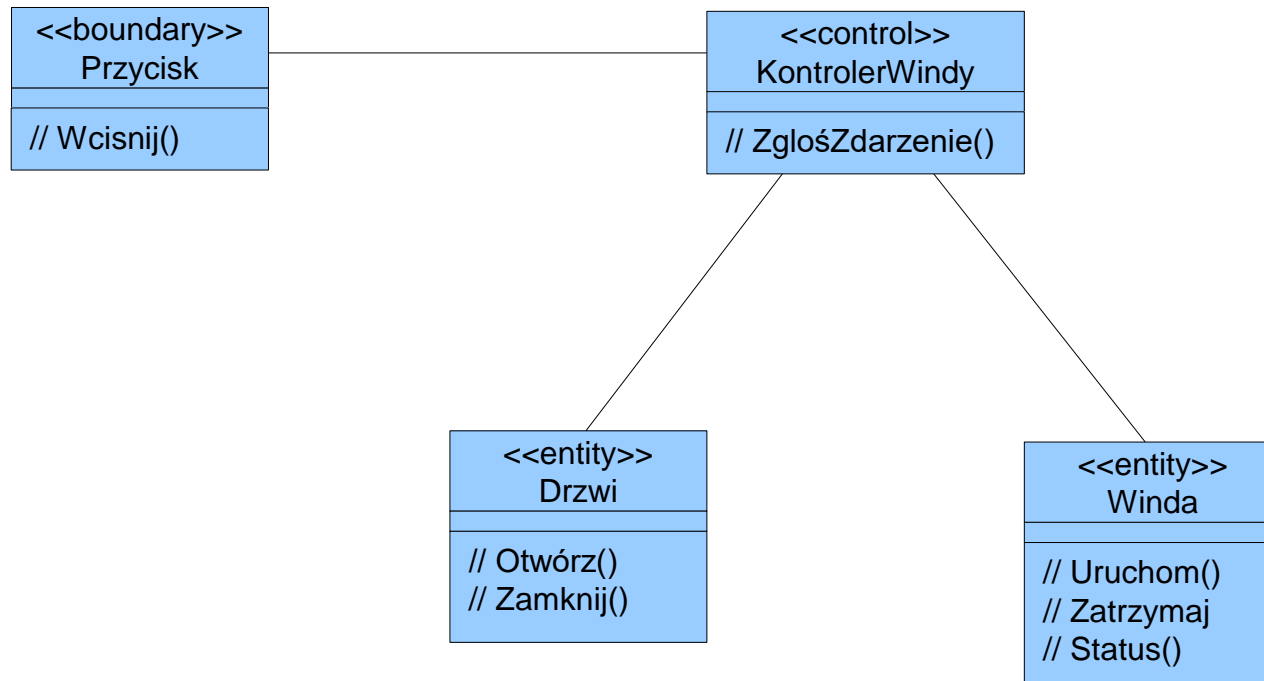
- Obiekt, adresat komunikatu musi go „rozumieć”, co oznacza, że klasa której jest wystąpieniem musi dostarczyć (definiować) tę operację.
- Konstruowanie diagramów interakcji może pomóc w identyfikowaniu zarówno asocjacji między klasami, jak i operacji w klasach, a przez to może prowadzić do korekty diagramu klas, i temu celowi zresztą głównie służy.

# Diagram komunikacji a diagram klas



# Diagram klas (VOPC View Of Participating Classes)

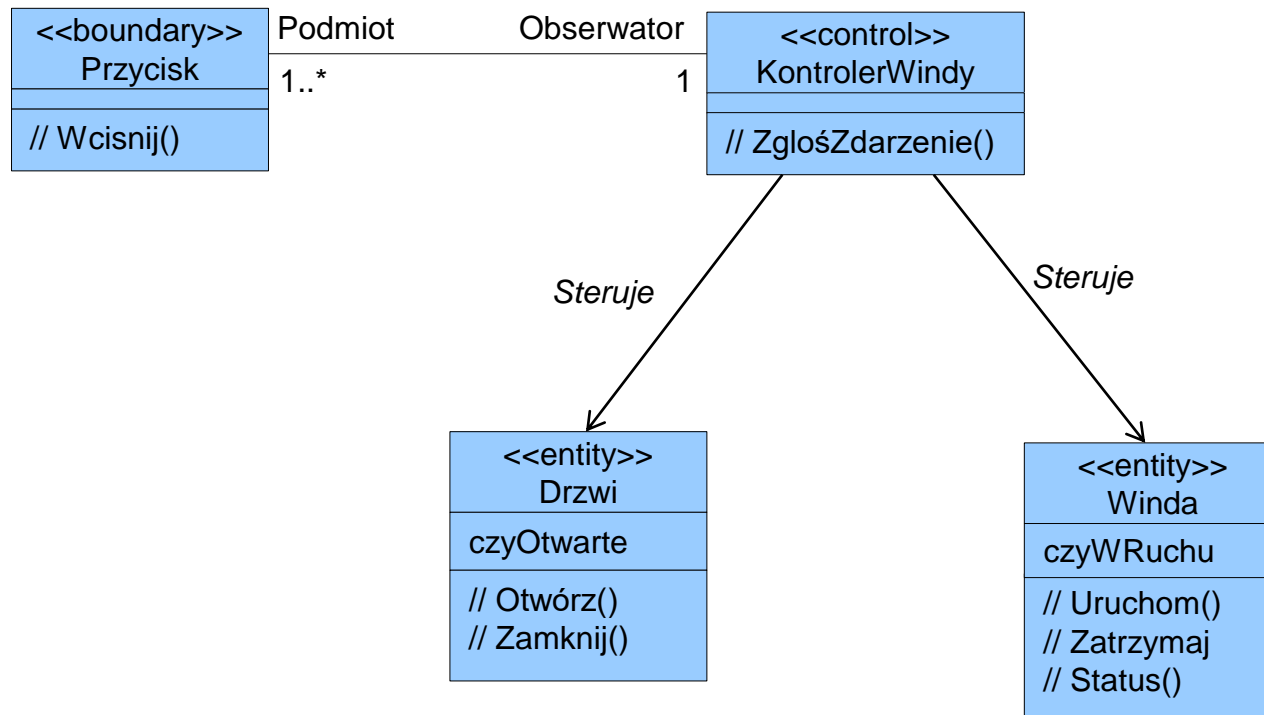
Diagram klas zawierający klasy, które współdziałają w wykonaniu określonego Przypadku Użycia



# Krok 2.2: Opisanie atrybutów i asocjacji

- Atrybuty
  - przechowują informacje
  - Są atomowe, nie posiadają odpowiedzialności
- Asocjacje
- Relacje pomiędzy dwoma (lub większą liczbą klas)
  - Asocjacja
  - Agregacja
  - Generalizacja

# Przykład diagramu klas z powiązaniem i atrybutami



# Krok 2.3: Przypisanie mechanizmów analizy

- Mechanizm analizy (konceptyjny):
- Opisuje podstawowe aspekty rozwiązania w sposób niezależny od konkretnej implementacji.
- Używany jest podczas analizy w celu zmniejszenia złożoności problemu na tym etapie.
- Pewne złożone zachowanie opisujemy określonym stereotypem nie przejmując się szczegółami implementacyjnymi.

Własności mechanizmu analizy opisują нефunkcjonalne wymagania związane z systemem

# Mechanizm analizy – przykłady

- Trwałość (ang. *persistence*)
- Typ komunikacji (IPC, RPC)
- Rozproszenie (ang. *distribution*)
- Zarządzanie transakcjami (ang. *transaction management*)
- Synchronizacja (ang. *process synchronization*)
- Bezpieczeństwo (ang. *security*)
- Obsługa błędów (ang. *error detection, handling, reporting*)
- Nadmiarowość (ang. *redundancy*)
- Interfejsy spadkowe (ang. *legacy interface*)



# Właściwości mechanizmu analizy

## - przykłady

- Trwałość
- Ziarnistość
- Natężenie
- Czas trwania
- Mechanizm dostępu
- Częstotliwość dostępu
- Niezawodność
- Komunikacja międzyprocesowa
- Opóźnienie
- Synchronizacja
- Rozmiar komunikatu
- Protokół



# Właściwości mechanizmu analizy – przykłady cd.

## Interfejs spadkowy

- Opóźnienie
- Czas
- Mechanizm dostępu
- Częstotliwość dostępu

## Bezpieczeństwo

- Ziarnistość danych
- Ziarnistość użytkowników
- Reguły bezpieczeństwa
- Typy uprawnień

# Proces opisu mechanizmu analizy

Zbierz wszystkie mechanizmy analizy w postaci listy – takie same mechanizmy analizy mogą występować w różnych sytuacjach pod różnymi nazwami (trwałość, baza danych, repozytorium, składowanie) (komunikacja między-procesowa, zdalne wywołanie procedury, przekazywanie komunikatów)

Narysuj mapę powiązań pomiędzy klasami i mechanizmami analizy – klasy, nazwy mechanizmów, strzałki łączące klasy z odpowiednimi mechanizmami

Określ właściwości mechanizmów analizy – aby ułatwić w procesie projektowym wybór odpowiedniego rozwiązania projektowego należy opisać podstawowe właściwości mechanizmów.

Zamodeluj przy użyciu kolaboracji – kiedy mechanizmy zostały zidentyfikowane i nazwane, należy je zamodelować w postaci współpracujących ze sobą klas. Niektóre z tych klas nie udostępniają żadnej funkcjonalności specyficznej dla tworzonej aplikacji, ale jedynie służą wsparciu działania aplikacji (często umieszcza się je potem w specjalnej warstwie oprogramowania zwanej – middleware).

# Przykład:

- Klasa: **Terminarz Studenta**
- Mechanizm analizy: **Trwałość**
- Ziarnistość (rozmiar danych): 1 – 10 kb
- Objętość (liczba rekordów): do 2000
- Częstotliwość dostępu:
- Tworzenie: 500 dziennie
- Odczyt: 2000 na godzinę
- Aktualizacje: 1000 dziennie
- Usuwanie: 50 dziennie

# Krok 3: Unifikacja klas

- Dotychczas tworzyliśmy osobne modele dla każdego **przypadku użycia**.
- Teraz trzeba je zunifikować, tak aby wyodrębnić te, które mają unikatowa odpowiedzialność a połączyć te, których zadania są wspólne.
- Nazwa klasy analizy powinna odzwierciedlać istotę roli, jaką klasa gra w systemie. Nazwa musi być unikatowa.
- Należy pamiętać, że po uaktualnieniu klas należy uaktualnić opis realizacji przypadku użycia. Czasem uzupełnienie może dotyczyć oryginalnych wymagań (przypadków użycia) ale należy pamiętać o tym, że wymagania są częścią umowy z klientem i każda zmiana powinna być potwierdzona i kontrolowana.

# Krok 4: Kontrola poprawności modelu

## Klasy analizy:

- Czy klasy analizy mają sens?
- Czy nazwa klasy odzwierciedla rolę tej klasy w systemie?
- Czy klasa reprezentuje pojedynczą, dobrze zdefiniowaną abstrakcję?
- Czy atrybuty klasy oraz jej odpowiedzialności są ze sobą funkcjonalnie powiązane?
- Czy klasa udostępnia wystarczającą funkcjonalność?
- Czy klasa spełnia wszystkie wymagania?



# Krok 5: Kontrola poprawności modelu cd.

Realizacja przypadków użycia:

- Czy wszystkie przepływy zostały zrealizowane (włącznie z alternatywnymi i wyjątkowymi)?
- Czy wszystkie obiekty zostały znalezione?
- Czy działanie wykonywanie w przepływach zostało jednoznacznie rozdzielone pomiędzy obiekty.
- Czy działanie zostało przypisane odpowiednim obiektom?
- Jeżeli dla przepływu zostało zdefiniowanych kilka diagramów interakcji to czy związki pomiędzy nimi są czytelne?



# Wykład 6

Projektowanie klas



# Projektowanie klas – kroki:

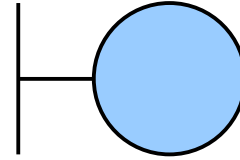
1. Utworzenie początkowego zestawu klas
2. Zdefiniowanie operacji
3. Zdefiniowanie metod
4. Zdefiniowanie stanów
5. Zdefiniowanie atrybutów
6. Zdefiniowanie zależności
7. Zdefiniowanie powiązań
8. Zdefiniowanie związku generalizacji – specjalizacji
9. Projektowanie wymagań нефunkcjonalnych

# Krok 1: Utworzenie początkowego zestawu klas

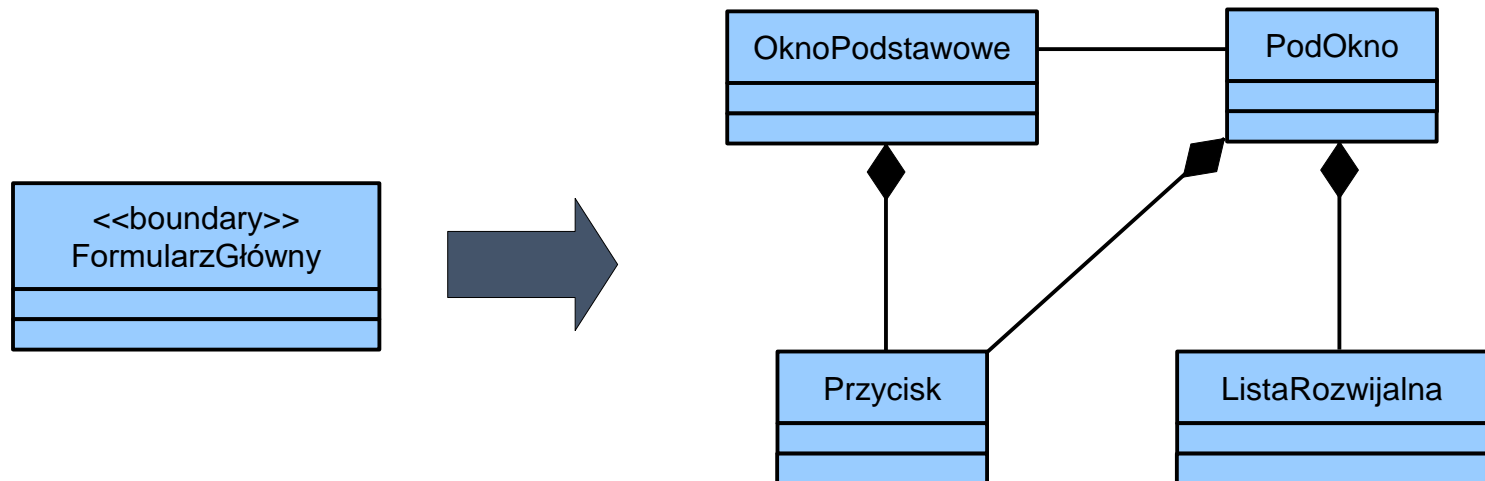
- Wstępem do procesu projektowania klas są **klasy analizy**.
- Celem procesu projektowania klas jest zmodyfikowanie tego zestawu (uzupełnienie, dokładniejsze opisanie, itp.)
- Co należy wziąć pod uwagę?
  - Jak klasy analizy będą realizowane w implementacji
  - Jakie wzorce projektowe wykorzystać, aby rozwiązać problemy implementacyjne
  - Jak będzie realizowany mechanizm analizy.

# Strategie projektowania klas

## Boundary:



- Klasy interfejsu użytkownika
  - Jakie narzędzia będą użyte do tworzenia UI?
  - Jaki procent interfejsu może być wygenerowany przez narzędzie?
- Klasy interfejsu systemowego
  - Zazwyczaj modelowane jako podsystem.

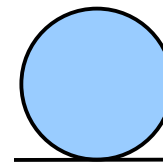


# Strategie projektowania klas

## Boundary:

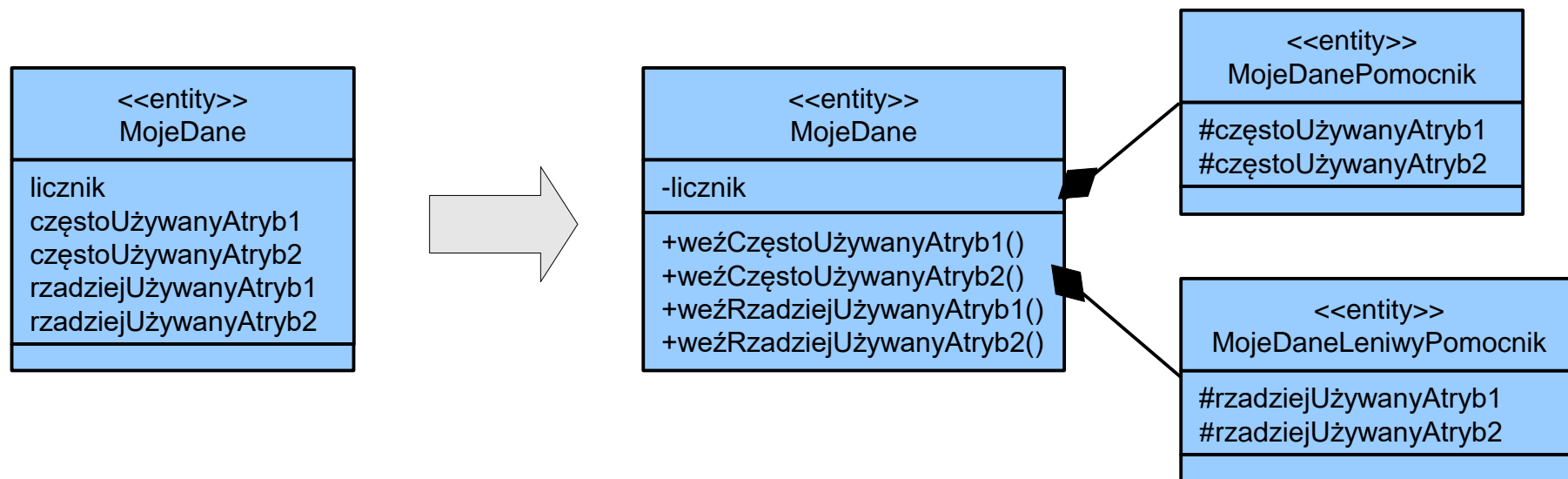
- Tworzenie graficznego interfejsu użytkownika może być rozpoczęte we wczesnych fazach projektu.
- Prototypy interfejsów użytkownika są często wykorzystywane w procesie definiowania przypadków użycia i mogą wydatnie zwiększyć efektywność technik pozyskiwania wymagań.

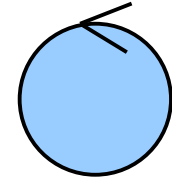
# Projektowanie klas Entity



Klasy Entity są zazwyczaj klasami pasywnymi i trwałymi. Kwestie wydajnościowe mogą wymusić modyfikację modelu projektowego.

Czasami dane muszą być przechowywane w różnych miejscach, co też powoduje potrzebę zmian modelu projektowego.





# Projektowanie klas Control

- Klasy Control wydzieliliśmy w procesie analizy przypadków użycia jako elementy hermetyzujące logikę sterowania realizacją przypadku użycia (scenariusza).
- Na etapie projektowania klas mogą one zostać usunięte, lub stać się pełnoprawnymi klasami projektowymi.

# Projektowanie klas Control, cd..

- Mogą zostać usunięte gdy ich zadaniem jest jedynie przekazywanie komunikatów z klas *Boundary* do klas *Entity*
- Stają się pełnoprawnymi klasami projektowymi gdy:
  - Hermetyzują znaczną ilość działania związanego z kontrolą przepływu.
  - Jest duże prawdopodobieństwo, że zadania, które wykonują mogą podlegać zmianom.
  - Zadania muszą być rozproszone pomiędzy wiele procesów (również zdalnych).
  - Zadania, które wykonują wymagają wprowadzenia mechanizmów zarządzania transakcjami.

# Zdefiniowanie operacji - Określanie dostępności metody

**Podstawowa zasada bezpieczeństwa!** Operacja powinna mieć dostępność minimalną z punktu widzenia wykonywanego zadania

Jeżeli na diagramach interakcji:

- Komunikat do obiektu jest wysyłany z zewnątrz – **dostęp publiczny (+ w UML)**
- Komunikat jest wysyłany z obiektu podklasy - **dostęp chroniony (# w UML)**
- Komunikat jest wysyłany od samego siebie – **dostęp prywatny (- w UML)**



# **Zdefiniowanie operacji – parametry operacji**

## **1. Parametry operacji**

- Czy są przekazywane przez wartość czy przez referencję?
- Czy są zmieniane przez operację?
- Czy są opcjonalne?
- Czy powinny mieć domyślne wartości?
- Czy mają zakres poprawnych wartości?

## **2. Im mniej parametrów tym lepiej**

## **3. Parametrami powinny być obiekty zamiast pojedynczych pól.**

# Inwarianty klasy - przykład

Student
nazwisko adres id <u>następnyWolnyId : int</u>
dodajPlan(plan : Plan, naSemestr : Semestr) pobierzPlan(naSemestr : Semestr) : Plan zalicz(przedmiot : Przedmiot) : boolean <u>pobierzNastępnyWolnyId() : int</u>

# Krok 3: Zdefiniowanie metod

## Operacja a metoda

**Operacja** jest funkcją, która może być zastosowana do obiektu. Operacja jest własnością klasy obiektów, ponieważ jest przechowywana w klasie.

*Przykład: możliwe operacje na obiektach klasy Firma:*  
zatrudnij,  
zwolnij,  
wypłacić dywidendę.

**Metoda** jest implementacją operacji w jednej z klas połączonych związkiem generalizacji-specjalizacji, co oznacza, że może być wiele metod implementujących daną operację.

# Krok 3: Zdefiniowanie metod

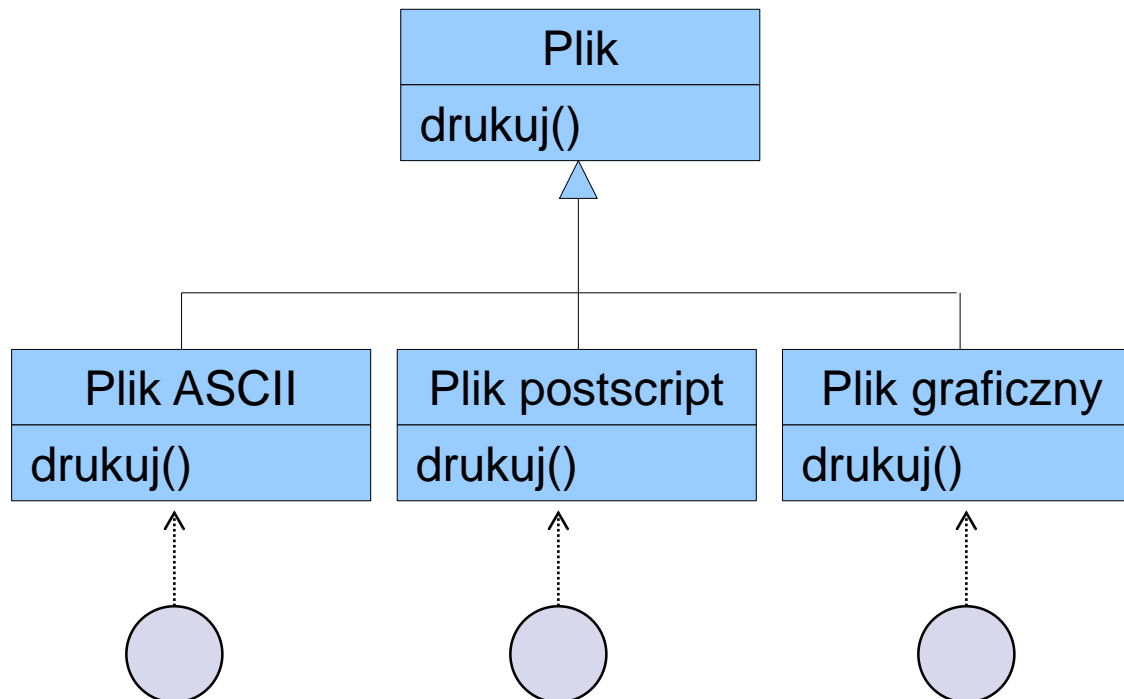
## Operacja a metoda

W kroku definiowania metody opisujemy implementację metody, czyli:

- Algorytm,
- Inne obiekty i operacje, które będą wykorzystywane,
- Sposób implementacji i wykorzystania parametrów i atrybutów wewnątrz metody,
- Sposób implementacji i wykorzystania powiązań pomiędzy klasami.

# Operacja a metoda cd...

Jedna operacja drukuj, ale różne sposoby drukowania. Trzy metody implementujące operację **drukuj**:



# Krok 4: Zdefiniowanie stanów

- Określenie w jaki sposób stan obiektu wpływa na jego zachowanie (wykonywanie operacji) i wyrażenie tych zależności na diagramach stanu.
- Dla obiektów, które przejawiają różnice w działaniu w zależności od stanu.
- Określenie stanów, przejść, identyfikacja zdarzeń.
- Powiązanie diagramów stanów z pozostałą częścią modelu.

# Diagramy stanu: maszyna stanu

Obiekt, w świetle swoich własności (unikatowa tożsamość, stan i zachowanie) może być traktowany jako automat o skończonej liczbie stanów, czyli pewną maszynę, która może znajdować się w danym momencie w jednym z wyróżnionych stanów, a także może oddziaływać na otoczenie i vice-versa.

# Diagramy stanu: maszyna stanu

- Maszyna stanu jest grafem skierowanym, reprezentowanym za pomocą notacji diagramów stanu, którego wierzchołki stanowią stany obiektu, a łuki opisują przejścia między stanami.
- Przejście między stanami jest odpowiedzią na zdarzenie.
- Zwykle, maszyna stanu jest przypisana do klasy i specyfikuje reakcje wystąpień danej klasy na zdarzenia, które do nich przychodzą, stanowiąc w ten sposób model historii życia (opis wszystkich możliwych stanów i przejść) dla obiektu danej klasy.

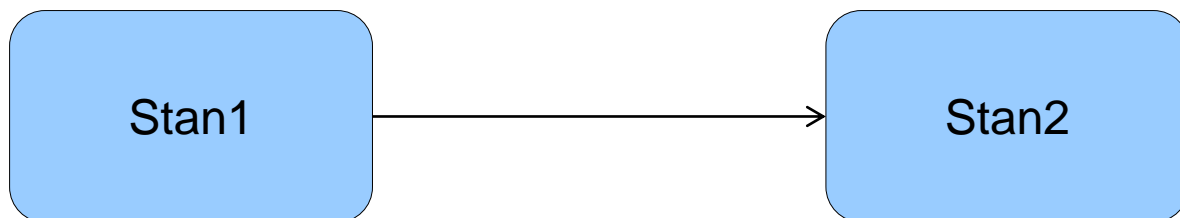


# Diagramy stanu: maszyna stanu

- Można przypisać maszynę stanu do przypadku(ów) użycia, operacji, kolaboracji, ale w tym znaczeniu - przepływu sterowania - częściej wykorzystuje się inne środki, np. diagramy aktywności.
- Takie podejście, separujące obiekt od reszty świata (innych obiektów w systemie czy poza nim), stanowiące podstawę do konstruowania diagramów stanu, pozwala na dokładną analizę zachowań pojedynczego obiektu, ale może nie być najlepszym sposobem na zrozumienie działania systemu jako całości.
- Diagramy dynamiczne najlepiej sprawdzają się w procesie analizy działania mechanizmów sterujących, takich jak np., interfejsy użytkownika czy sterowniki urządzeń.

# Diagramy stanu: maszyna stanu

- Graf skierowany, reprezentowanym za pomocą notacji diagramów stanu, którego wierzchołki stanowią **stany** obiektu, a łuki opisują przejścia między stanami
- Pozwala na dokładną analizę zachowań pojedynczego obiektu
- Wykorzystywane nie tylko w trakcie szczegółowego projektowania klas



# Stan obiektu c.d.

Stan, dotyczy pewnego fragmentu historii życia obiektu. Można go charakteryzować na trzy uzupełniające się sposoby:

- jako zbiór wartości obiektu (atrybutów i powiązań) w pewnym aspekcie podobnych (rozważane jest tu podobieństwo jakościowe),
- jako okres czasu, w którym obiekt oczekuje na zdarzenie,
- jako okres czasu, w którym obiekt przetwarza.

# Stan obiektu

Stan jest oznaczany za pomocą prostokąta z zaokrąglanymi rogami. Stan może mieć nazwę, ale często jest charakteryzowany jedynie poprzez wewnętrzne operacje.

Stan1

entry/ AkcjaWejściowa  
exit/ AkcjaWyjściowa  
do/ Aktywność

Wewnętrzne operacje:

**akcja** - operacja, której nie można przerwać (atomowa), powiązana z przejściem. Można założyć, że jest punktowa.

**lista akcji** - akcja1/akcja2/... - traktowana jest, jak pojedyncza operacja,

**aktywność** - operacja, którą można przerwać, powiązana ze stanem. Rozpoczyna się po wejściu w dany stan. Trwa pewien czas.

**lista aktywności** - podobnie, jak lista akcji,

**entry** - słowo kluczowe specyfikujące operacje, które zawsze są wykonywane na wejściu do stanu (rodzaj setup'u),

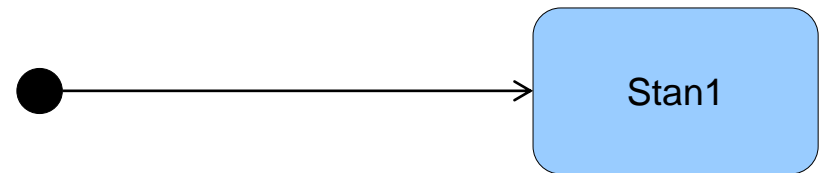
**exit** - operacje zawsze wykonywane na wyjściu (rodzaj porządkowania "po"),

**do** - operacje wykonywane w trakcie znajdowania się w danym stanie.

# Stany specjalne

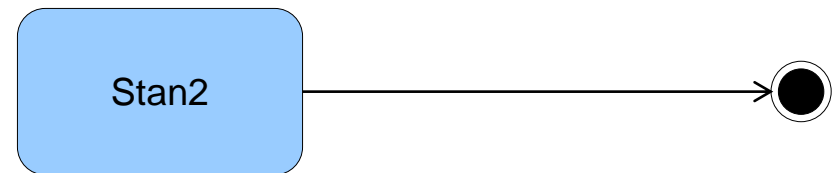
## Stan początkowy:

- stan w którym znajduje się obiekt po utworzeniu
- obowiązkowy i może być tylko jeden (1)



## Stan końcowy:

- koniec życia obiektu
- opcjonalny
- może być ich więcej niż jeden (0..\*)



# Identyfikacja stanów

Stany obiektu zazwyczaj są wyznaczane przez:

- Atrybuty, które zmieniając swoje wartości znacząco wpływają na przetwarzanie zadań przez ten obiekt.
- Atrybuty liczbowe, mające zdefiniowane zakresy
- Atrybuty logiczne
- Powiązania a dokładniej ich obecność lub brak

Oprócz zidentyfikowania stanów należy również zdefiniować co oznacza, że obiekt znajduje się w danym stanie

# Zdarzenia

- Zdarzeniem jest coś, co następuje w jednym punkcie czasowym (z perspektywy naszej percepcji czasu) i warto jest analizowania z punktu widzenia celów projektowanego systemu.
- Samo zdarzenie nie trwa w czasie, ale fakt zaistnienia zdarzenia jest rejestrowany i trwa aż do momentu, gdy jakiś podmiot go „skonsumuje” ( innymi słowy zdarzenie nie musi być obsłużone od razu w momencie wystąpienia - może być wpisane na listę zdarzeń oczekujących na obsługę).
- Wszystko, co wywołuje pewne skutki w systemie może być modelowane jako zdarzenie.

# Zdarzenia

Zdarzenia mogą być:

uporządkowane w czasie (synchroniczne), np. odlot samolotu z Warszawy i przylot tego samolotu do Paryża, ale możemy także rozpatrywać pewne zdarzenia jako współbieżne (asynchroniczne), np. naciśnięcie klawisza myszy i odlot samolotu są zdarzeniami wzajemnie niezależnymi i mogą być rozpatrywane jako współbieżne.

Zdarzenie w sensie opisu pewnego zjawiska jest klasyfikatorem i jako klasyfikator może posiadać atrybuty, np. zdarzenie odlot samolotu może mieć datę i godz. odlotu jako swoje atrybuty, co zapisujemy następująco: odlot samolotu (data, godz.). Wystąpienie zdarzenia jest odlotem z ustalonymi, konkretnymi wartościami obu atrybutów.



# Zdarzenia - podsumowanie

- Zdarzenie jest czymś co zachodzi w punkcie czasowym i jest istotne z punktu widzenia projektowanego systemu. Może być synchroniczne i asynchroniczne:
  - Naciśnięcie przez użytkownika klawisza myszy
  - Zmiana wartości atrybutu obiektu
  - Odlot samolotu z lotniska
- Zdarzenie może posiadać atrybuty:
  - Odlot samolotu: data, godzina

# Zdarzenia – typy

Typ zdarzenia	Opis	Składnia
<b>wołanie</b>	otrzymanie przez obiekt synchronicznego żądania wykonania operacji - najbardziej podstawowy rodzaj zdarzenia	<b>op</b> (a : T)
<b>zmiana</b>	spełnienie warunku typu Boolean, np. when (x =10); zdarzenie typu <b>zmiana</b> jest użyteczne np. do modelowania sytuacji, gdy obiekt zmienia stan po otrzymaniu odpowiedzi na wysłany przez siebie komunikat	<b>when</b> (wyrażenie)
<b>sygnał</b>	otrzymania przez obiekt asynchronicznego żądania wykonania operacji; użyteczne do modelowania zdarzeń przychodzących z zewnątrz systemu	<b>nazwa_syg</b> (a : T)
<b>czas</b>	upłynięcie czasu określonego w sposób bezwzględny lub względny, np. after (5 sec.)	<b>after</b> (czas) <sup>82</sup>

# Identyfikacja zdarzeń

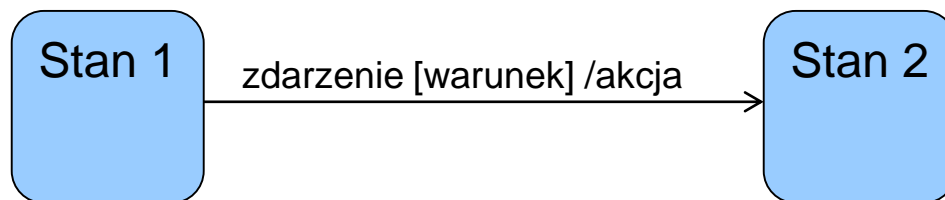
- Operacje, będące interfejsem obiektu
- Operacja Uruchom dla obiektu windy
- Operacja DodajProfesora dla obiektu reprezentującego kurs na uczelni.
- Obiekt musi odpowiadać na wszystkie komunikaty, które przychodzą do niego na wszystkich diagramach interakcji.

# Przejście

W ogólności, przejście może być opisane przez zdarzenie, które je wywołało, warunek oraz akcję (akcje), która jest wykonywana przed ewentualną zmianą stanu.

przejście

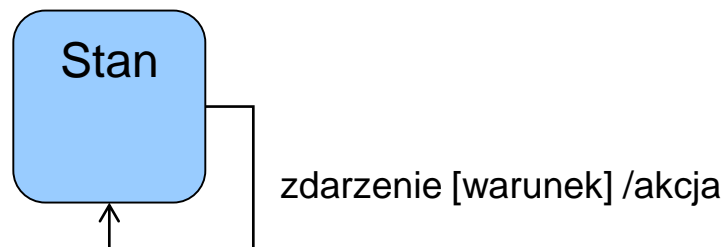
**przejście zewnętrzne**  
(*external transition*)



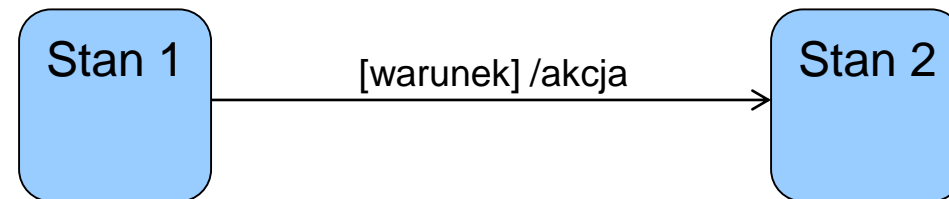
**przejście wewnętrzne**  
(*internal transition*)



**samo-przejście**  
(*selftransition*)



**przejście automatyczne**  
(*completion transition*)



# Przejście

- **Przejście zewnętrzne** – wykonywane są akcje wyspecyfikowane po słowie kluczowym **exit** s stanie 1 oraz akcje wyspecyfikowane po słowie kluczowym **entry** w stanie 2.
- **Przejście wewnętrzne** – nie ma zmiany stanu nie wykonywane są akcje **exit** i **entry**.
- **Samo-przejście** - w przeciwieństwie do przejścia wewnętrznego, przy wychodzeniu ze stanu wykonywane są wszystkie akcje wyspecyfikowane po słowie kluczowym **exit**, podobnie - przy ponownym wchodzeniu do stanu - są wykonywane akcje wyspecyfikowane po słowie kluczowym **entry**.
- **Przejście automatyczne** – aktywność wyspecyfikowana po słowie kluczowym **do** zakończyła się, oraz wykonywane są akcje wyspecyfikowane po słowie kluczowym **exit** s stanie 1 oraz akcje wyspecyfikowane po słowie kluczowym **entry** w stanie 2.

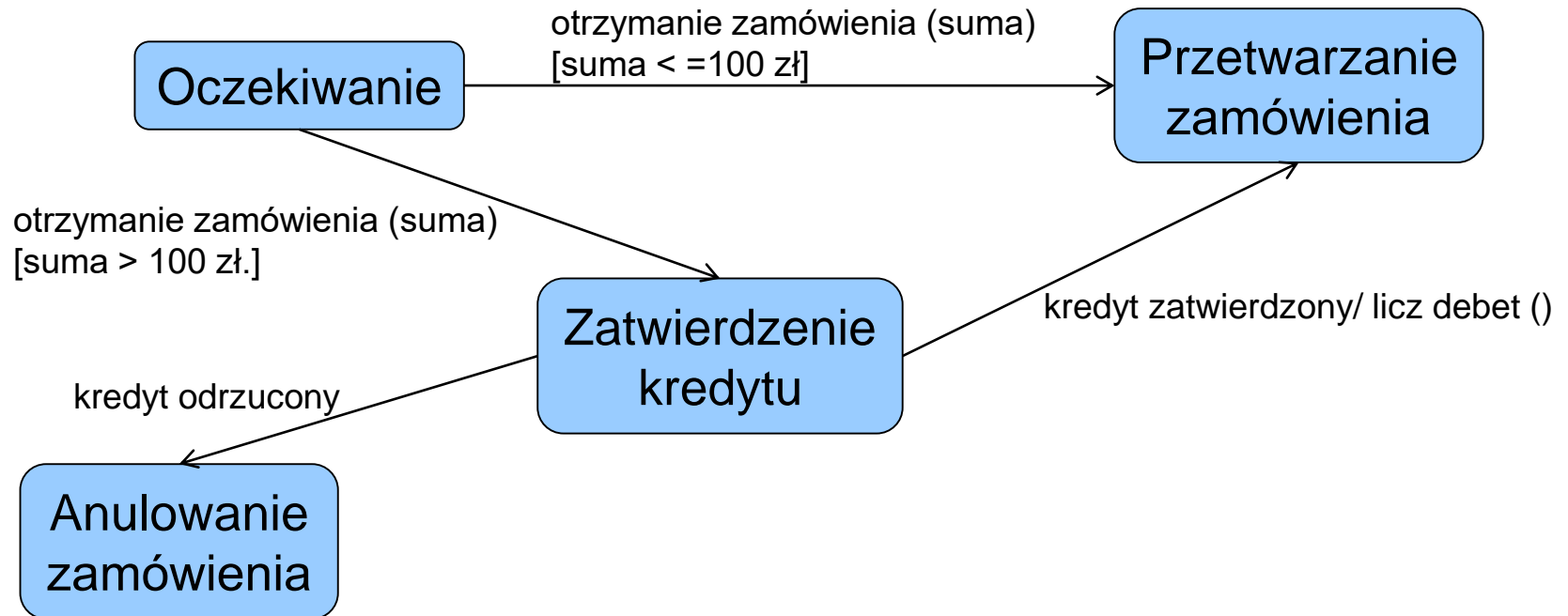
# Przejście

- Warunek typu Boolean, występujący w specyfikacji przejścia, może dotyczyć zarówno atrybutów maszyny stanu, jak i argumentów zdarzenia, które odpaliło dane przejście.
- Warunek podlega oszacowaniu tylko raz, w momencie wystąpienia zdarzenia.
- Jeśli warunek przyjmie wartość TRUE - przejście będzie miało miejsce.
- Jeśli nie wszystkie możliwości zostały zrealizowane, zdarzenie zostanie zignorowane.

# Identyfikacja przejść – zdarzenie[warunek]/akcja

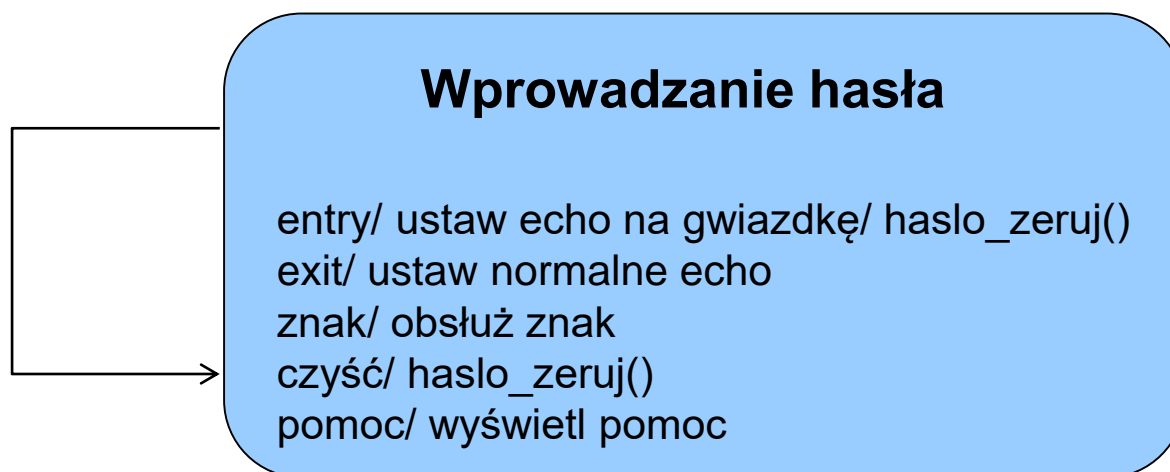
- Dla każdego stanu należy określić jakie zdarzenie powoduje przejście i do jakiego stanu. W razie potrzeby należy dołączyć dodatkowe warunki przejścia (jedno zdarzenie w zależności od warunku może powodować przejście do kilku stanów)
- Zdarzenia wciśnięcia przycisku przez pasażera w windzie.
- Przejście do stanu Zatrzymania pod warunkiem, że został wciśnięty przycisk STOP.

# Przykłady przejść (zewnętrzne)





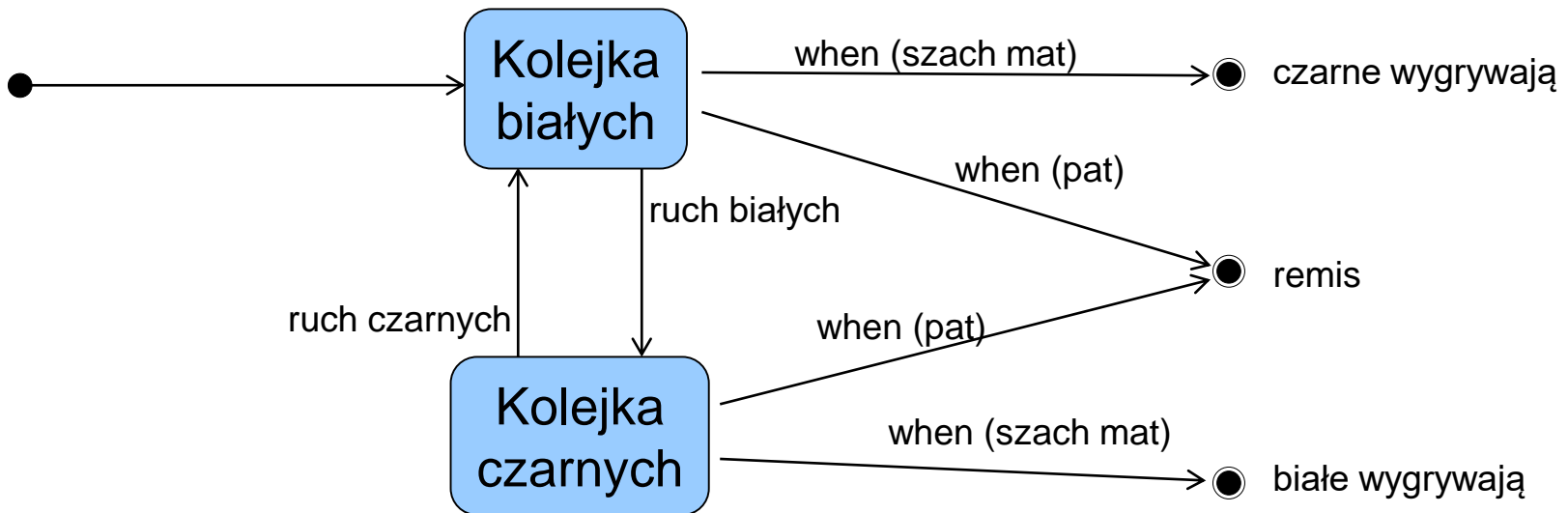
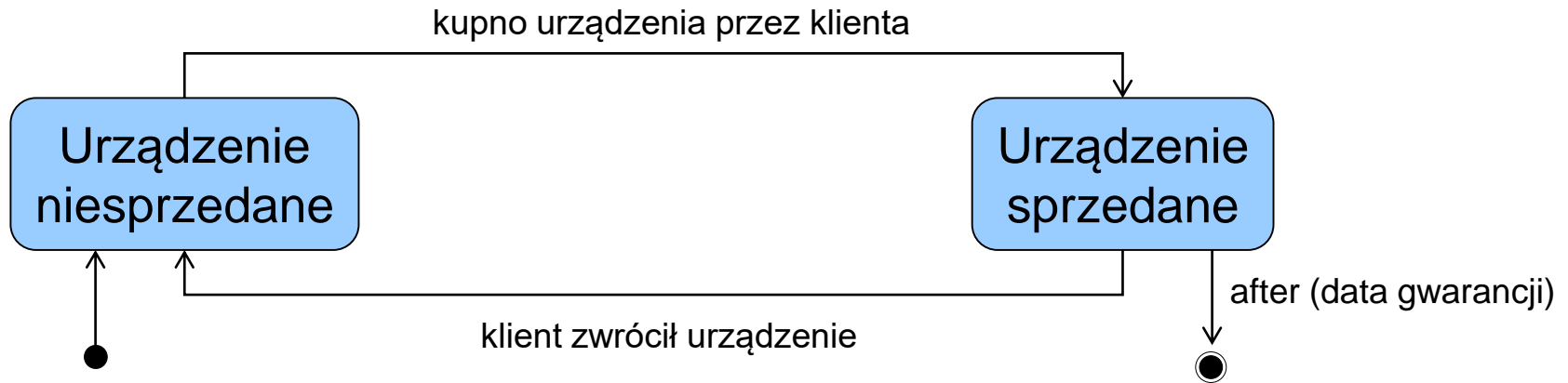
# Przykłady przejść (wewnętrzne)



# zdarzenie[warunek]/akcja

Rodzaj akcji	Opis	Składnia
przypisanie ( <i>assignment</i> )	przypisanie wartości do zmiennej	zmienna := wyrażenie
wołanie ( <i>call</i> )	wywołanie operacji na obiekcie; czeka się na zakończenie operacji; może być zwracana wartość	nazwa_op (arg, ...)
nowy ( <i>create</i> )	utworzenie nowego obiektu	nowy nazwa_klasy (arg, ...)
usuń ( <i>destroy</i> )	usunięcie obiektu	usuń ()
wyślij ( <i>send</i> )	utworzenie wystąpienia sygnału i wysłanie do obiektu (ów)	nazwa_sygnału (arg, ...)
zakończ ( <i>terminate</i> )	samodestrukcja obiektu	zakończ
powrót ( <i>return</i> )	specyfikuje instrukcję powrotu	powrót wartość_zwracana

# Przykłady

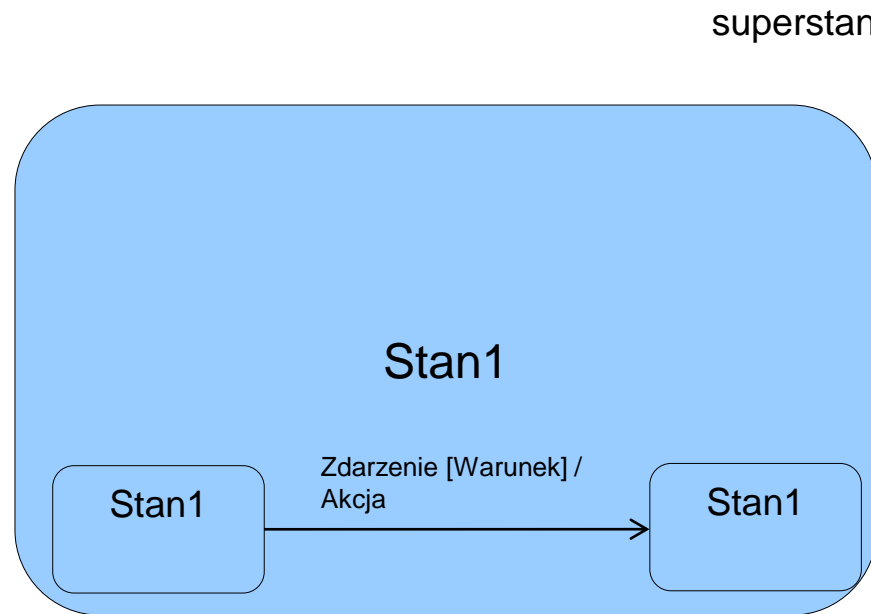


# Stany – informacje dodatkowe

- Stan prosty nie posiada substruktury, jest specyfikowany przez zbiór akcji (aktywności) oraz przejść.
- Stan złożony może być zdekomponowany na stany bardziej proste; dekompozycja jest tu rodzajem specjalizacji:
- Każdy z podstanów dziedziczy przejścia nadstanu.
- Tylko jeden z podstanów może być aktywny w danym momencie.
- Generalizacja stanów jest formą zagnieżdżania stanów.

# Stany – informacje dodatkowe

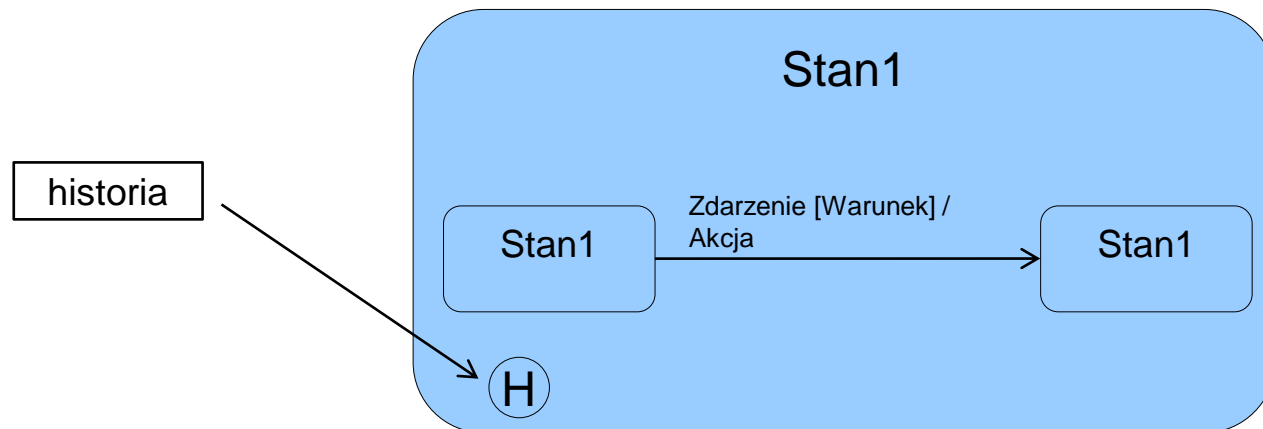
Diagramy stanów mogą być poziomowane. Przy dużej złożoności na diagramie wyższego poziomu można wyrażać tzw. superstan, który składa się z podstanów.



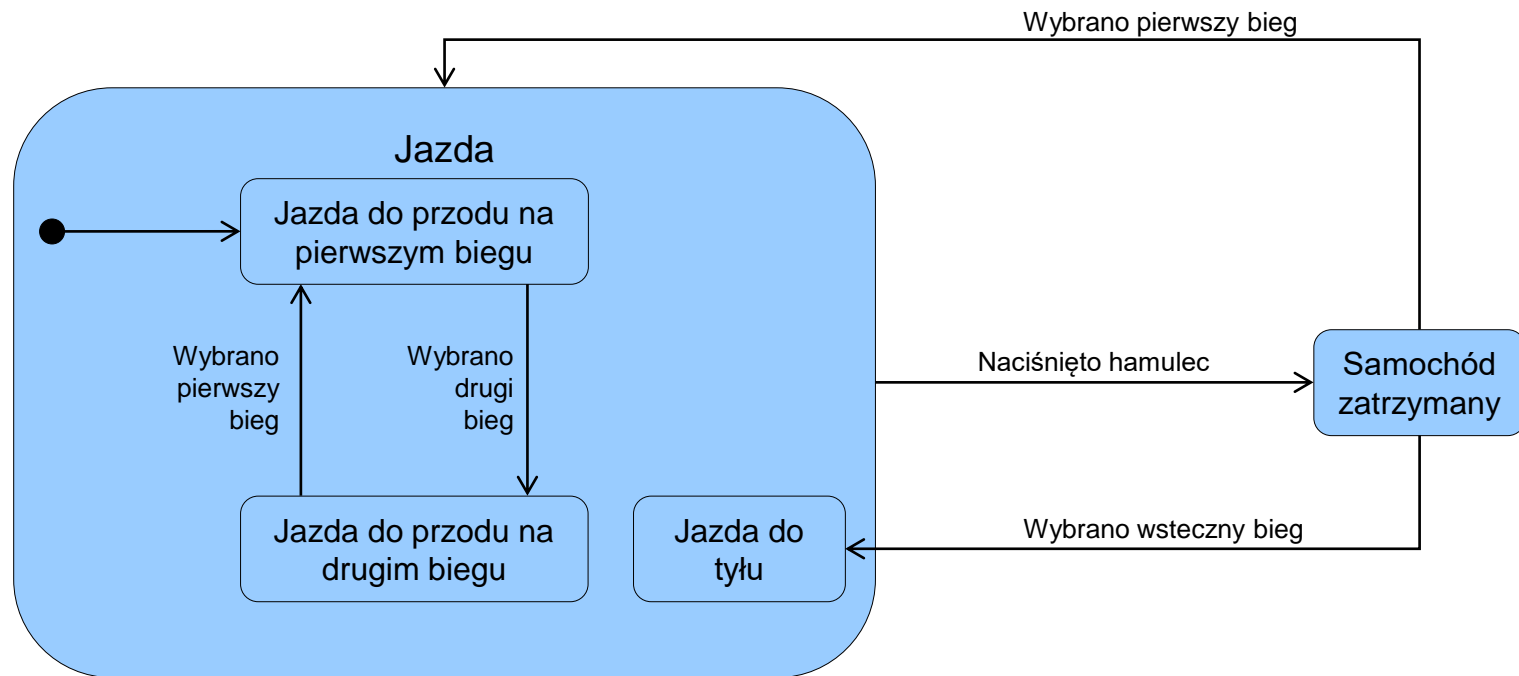
# Stany – informacje dodatkowe

## cd.

- Historia stanów – wsparcie dla modelowania sytuacji, w których musimy zapamiętywać stan w którym obiekt znajdował się przed ostatnim przejściem.
- W momencie gdy obiekt wychodzi z danego superstanu, zapamiętywany jest podstan, w którym się znajdował. Jeżeli obiekt powróci do danego superstanu powinien znaleźć się w zapamiętanym podstanie.
- Jeżeli stan nie jest zapamiętywany obiekt znajdzie się w stanie oznaczonym jako początkowy.



# Stany złożone: przykład



# Powiązanie stanów z pozostałą częścią modelu

- Zdarzenia mogą stać się operacjami obiektów
- Definicję metod muszą być uzupełnione o informacje związane ze stanem
- Stany są często opisywane przy użyciu atrybutów





# Krok 5: Zdefiniowanie atrybutów

## Atrybuty opisują:

- Informacje, które muszą być przechowywane i zarządzane przez klasę
- Informacje potrzebne metodom do przetwarzania
- Informacje o stanie

## Sygnatura atrybutu:

stereotyp dostępność nazwa\_atrybutu : typ = wartość\_domyślna

# Atrybuty

Atrybuty mogą być:

- **proste**: imię, nazwisko, nazwisko panieńskie, wiek, płeć, stosunek do służby wojskowej
- **złożone**: data ur. , adresy, lista poprz. miejsc pracy, zdjęcie
- **opcjonalne**: stosunek do służby wojsk, nazwisko panieńskie
- **powtarzalne**: lista poprz. miejsc pracy
- **pochodne**: wiek
- **instancją klasy**: adres firmy
- **obiektem**: zdjęcie

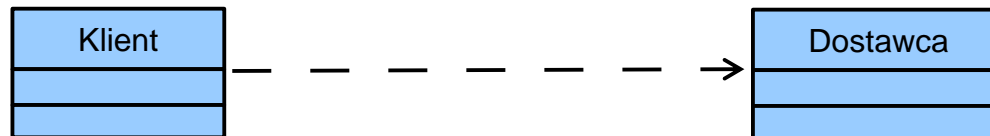
Pracownik
imię nazwisko nazwisko panieńskie data ur. wiek adres zamieszkania płeć stosunek do służby wojsk. [0..1] lista poprz. miejsc pracy [0..*] <u>adres firmy</u> zdjęcie

# Atrybuty

- Atrybuty klasy należą do inwariantów danej klasy.
- Kiedy z atrybutu warto zrobić klasę?
- W jakiej sytuacji atrybut adres firmy przestanie być atrybutem klasy?

# Krok 6: Zdefiniowanie zależności

- Związek zależności jest związkiem niestukturalnym (w przeciwieństwie do asocjacji czy agregacji)
- Podczas analizy zakładaliśmy, że wszystkie związki są strukturalne. Teraz musimy zdecydować o typie ścieżki komunikacji występującej pomiędzy obiektami.



# Zależności a asocjacje (1)

**Zależność** - jeżeli klient widzi dostawcę poprzez:

- Globalną referencję (obiekt dostawcy jest globalny)
- Parametr operacji (obiekt dostawcy jest parametrem metody klienta lub typ dostawcy jest typem zwracanym przez metodę klienta)
- Zmienną lokalną (obiekt dostawcy jest tymczasową zmienną operacji w klasie klienta)

**Asocjacja** - jeżeli obiekt dostawcy jest atrybutem klasy klienta (poprzez referencję, wartość).

Należy przejrzeć asocjacje pod kątem możliwości ich zamiany na związek zależności

# Zależności a asocjacje (2)

- Na diagramach współpracy
- Instancją asocjacji jest powiązanie
- Wszystkie powiązania stają się asocjacjami, chyba, że są globalne, lokalne lub są parametrem (operacji)
- Zależności są „przezroczystymi” powiązaniami
- Mają ograniczony czas życia
- Są niezależne od kontekstu
- Są ogólne (niewiele nam mówią)

# Zmienne lokalne

op1() w klasie ClassA zawiera lokalną zmienną typu ClassB

Diagram klas

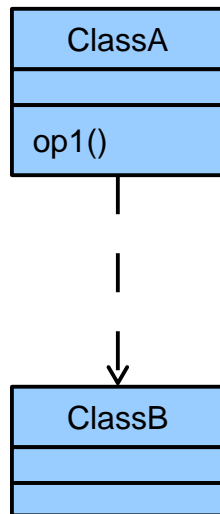
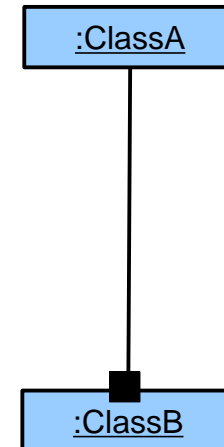


Diagram komunikacji



# Parametr

Obiekt klasy ClassB jest przekazywany do obiektu klasy ClassA jako parametr operacji op(1)

Diagram klas

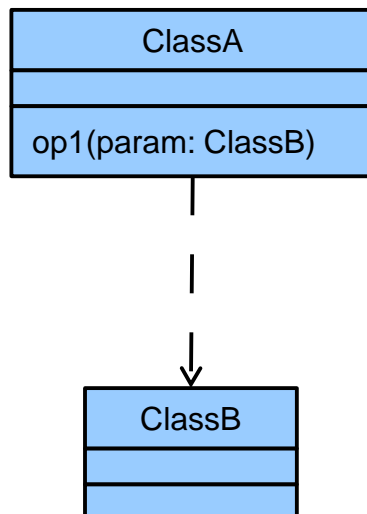
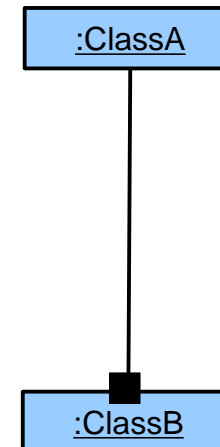


Diagram komunikacji





# Globalna referencja

Obiekt klasy ClassUtility jest widoczny, ponieważ jest globalny

Diagram klas

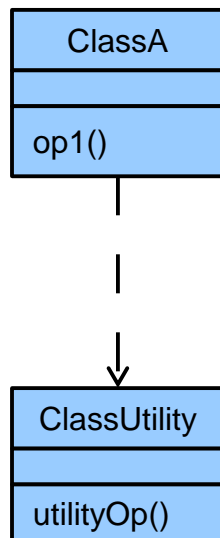
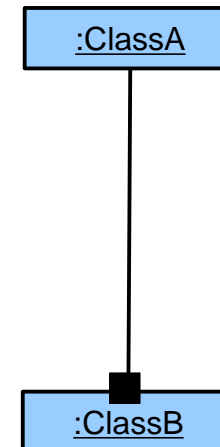


Diagram komunikacji



# Krok 7: Zdefiniowanie (projektowanie) asocjacji

Pozostałe asocjacje (które nie zostały zamienione na zależności) należy teraz doprecyzować poprzez określenie:

- Agregacji
- Kompozycji
- Nawigacji
- Klas asocjacji
- Liczności

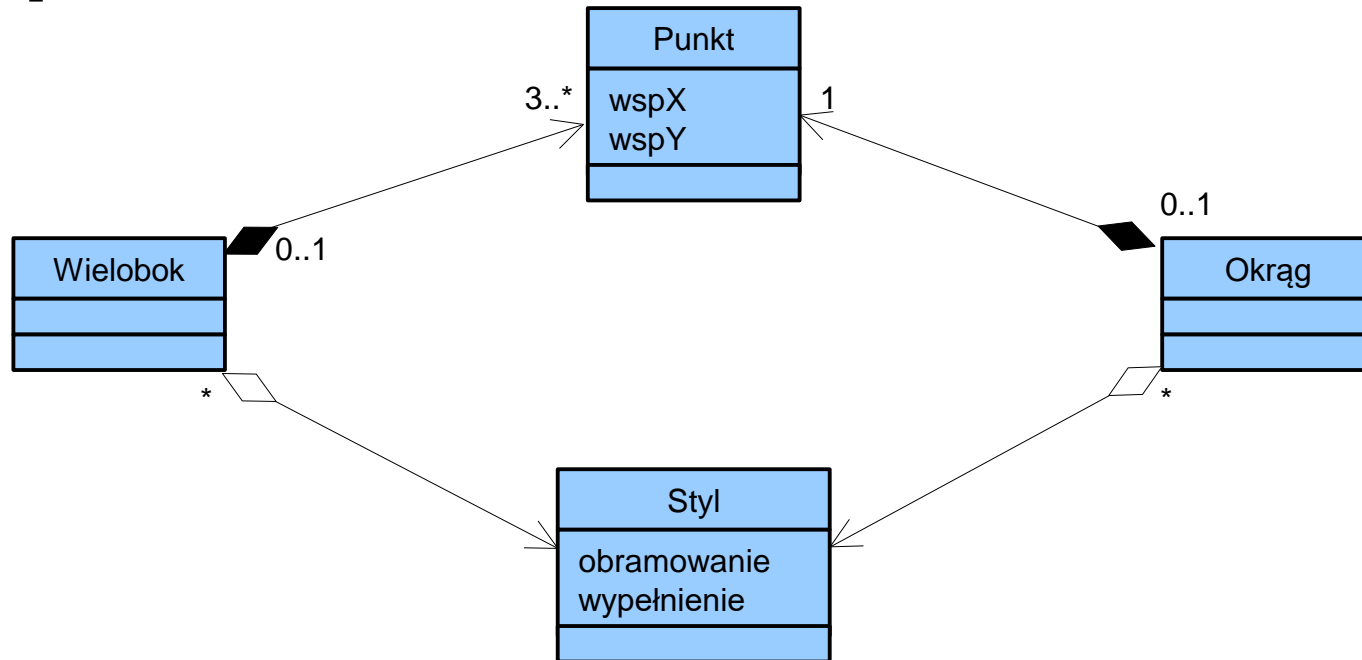
# Agregacja i kompozycja

Projektując agregacje musimy się zastanowić:

- Które asocjacje zamienić na agregacje?
- Które agregacje powinny stać się kompozycjami?



# Agregacja a kompozycja przypomnienie



Kompozycję należy użyć w momencie, gdy istnieje silna zależność pomiędzy całością a częścią (definicja całości jest niekompletna bez części).

Kompozycja powinna być zdefiniowana, jeżeli czas życia całości i części jest skorelowany.

# Kompozycja a atrybuty

Wykorzystaj kompozycję gdy:

- Elementy klasy muszą posiadać tożsamość
- Wiele klas ma te same własności
- Własności klasy mają złożoną strukturę (również posiadają własności)
- Własności klasy mają zachowanie (posiadają operacje)
- Własności klasy są elementami asocjacji
- W przeciwnym wypadku użyj atrybutów

# Nawigacja (ang. navigability) – asocjacje skierowane

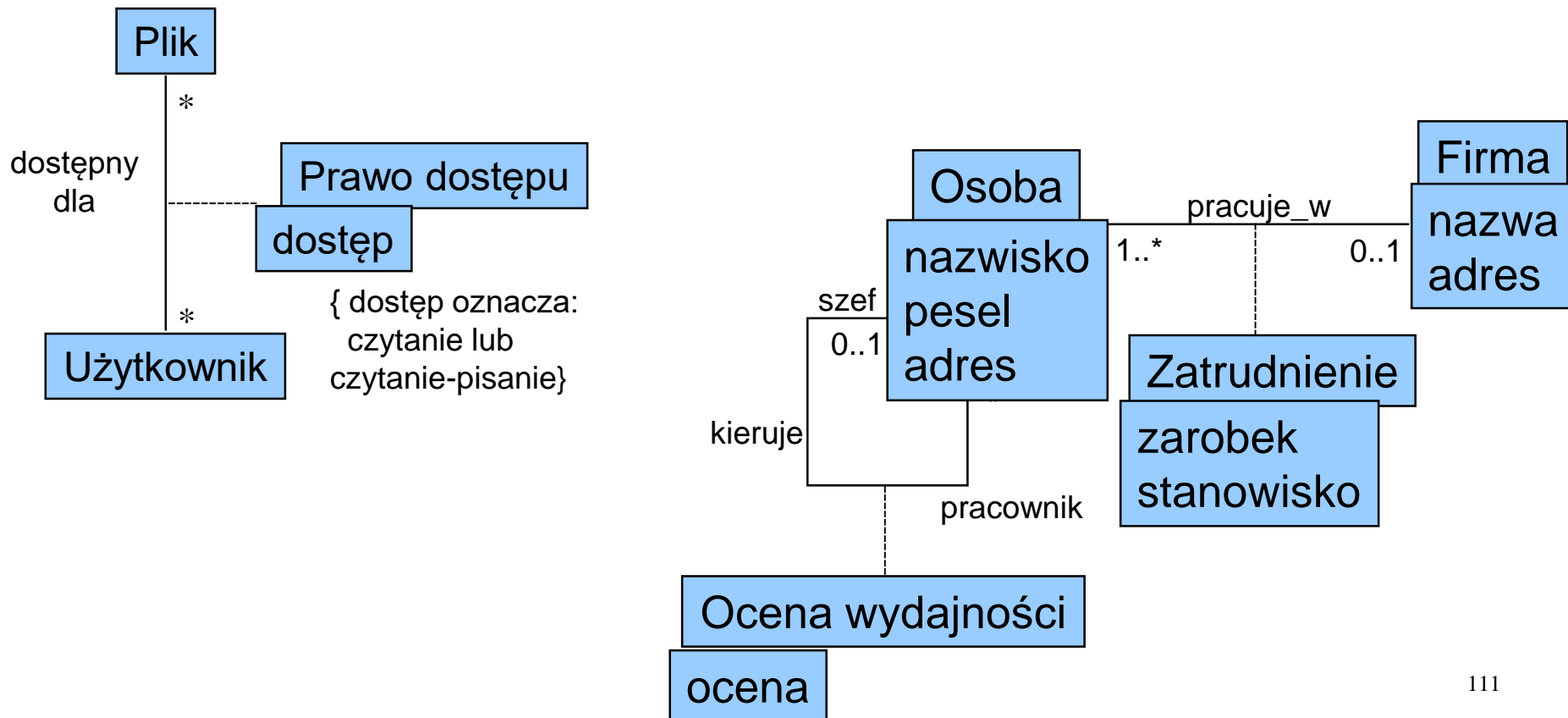
- Opisuje sytuację, w której wymagane jest aby z poziomu jednej klasy była możliwa nawigacja do drugiej, przy wykorzystaniu asocjacji.
- Nawigacja może być implementowana na wiele sposobów: referencję do obiektu, powiązaną tablicę, tablicę haszującą, lub inną technikę umożliwiającą odniesienie się jednego obiektu do drugiego.
- W języku UML asocjacje są domyślnie nawigowalne w obu kierunkach.



Należy określić, które kierunki są rzeczywiście potrzebne

# Klasa asocjacji

Jeżeli asocjacja ma atrybuty należy stworzyć z nich nową klasę  
(zależność wiele – do – wielu)

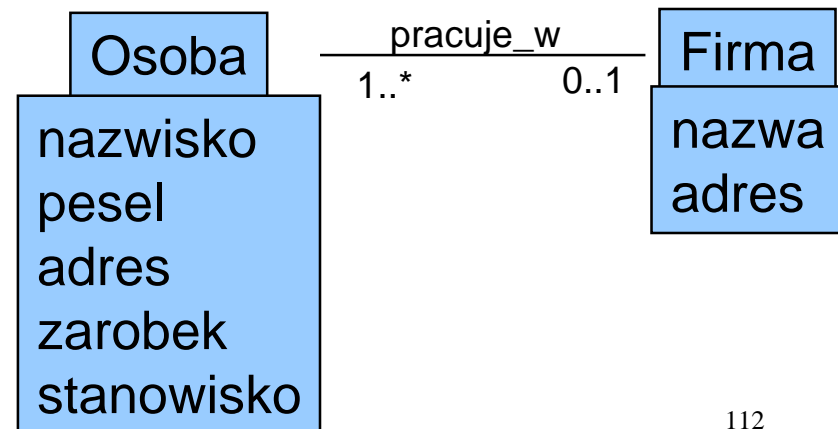
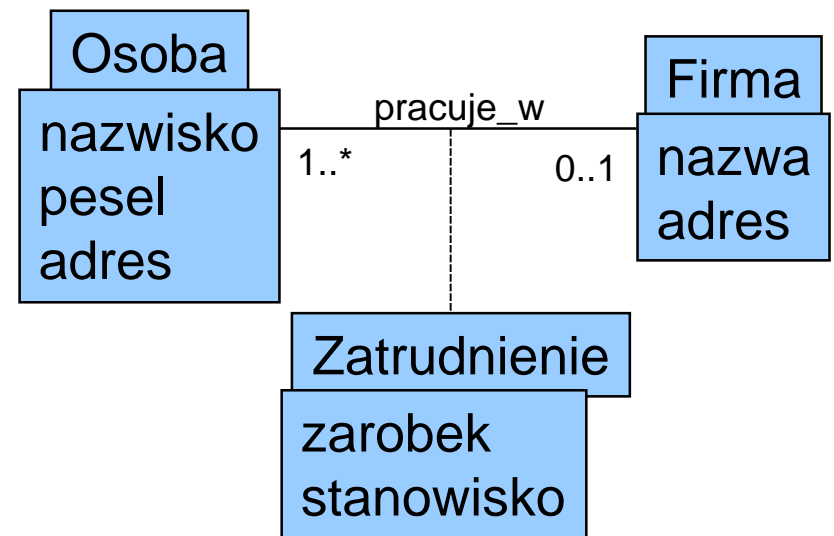


# Klasy asocjacji c.d.

**Zalecane jest**, by przypisywać do klasy tylko te atrybuty, które są dla tej klasy stabilne.

Eksperyment myślowy: co będzie, jeżeli liczność asocjacji się zmieni? Dość często okazuje się wtedy, że atrybut jest atrybutem asocjacji, a nie klasy.

**Forma nie zalecana**, mniej elastyczna: np. po zmianie powiązania na wiele-do-wielu trzeba zmieniać położenie atrybutów.





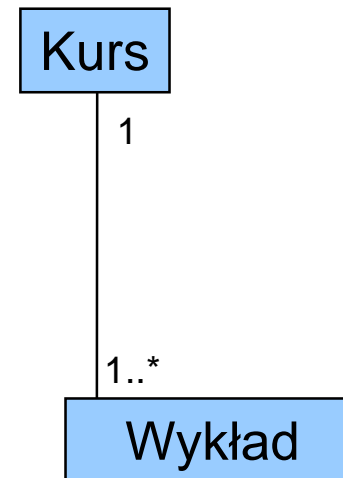
# Projektowanie liczności

## Liczność 1 lub 0..1

- Może być zaimplementowana bezpośrednio jako wartość lub wskaźnik
- Nie wymagane jest żadne dodatkowe projektowanie

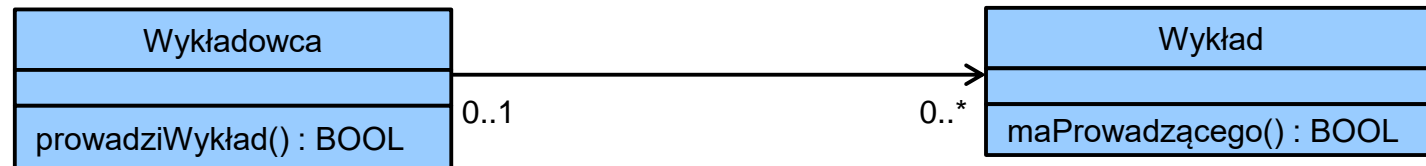
## Liczność > 1

- Dodatkowe projektowanie może być konieczne
- Wymagana jest kolekcja obiektów



# Projektowanie liczności – powiązania opcjonalne

Jeżeli powiązanie jest opcjonalne, należy pamiętać o tym, aby dodać operację pozwalającą na testowanie istnienia powiązania



# Krok 8: Zdefiniowanie związku generalizacji-specjalizacji

Związek generalizacji może występować:

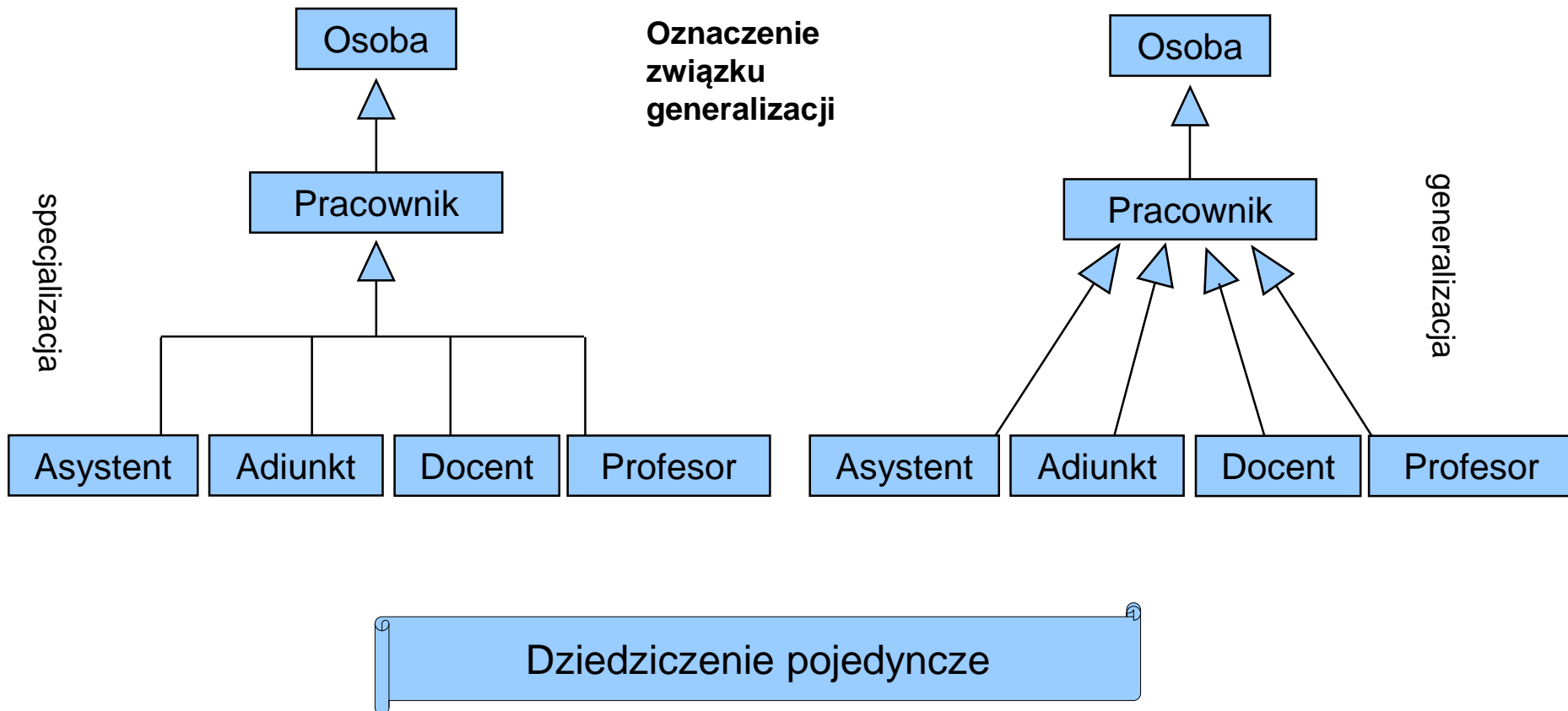
- Jako nieodłączny element dziedziny problemowej (wykrywany zazwyczaj w fazie analizy)
- Jako element uproszczenia implementacji (odkrywany w fazie projektowania)

# Generalizacja – specjalizacja - przypomnienie

**Generalizacja** – nazwa związku

**Dziedziczenie** – mechanizm modelujący związek  
generalizacji

# Generalizacja – specjalizacja - przypomnienie



# Ograniczenia dziedziczenia - przypomnienie

## Complete (domyślne)

- Koniec drzewa dziedziczenia, klasa, po której nie można dziedziczyć (na diagramie pokazano wszystkie klasy potomne – więcej nie będzie)

## Incomplete

- Drzewo dziedziczenia może ulec rozbudowie (nie wszystkie klasy potomne zostały pominięte na diagramie)

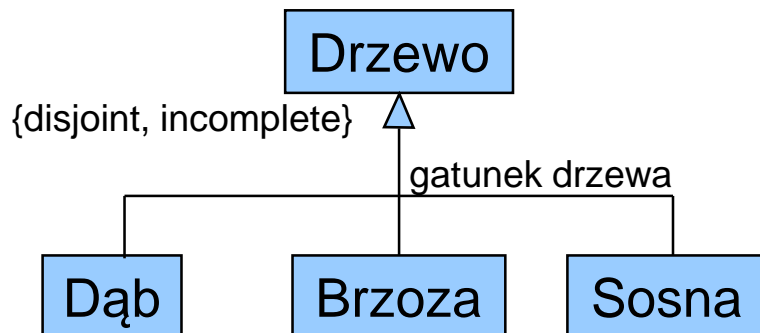
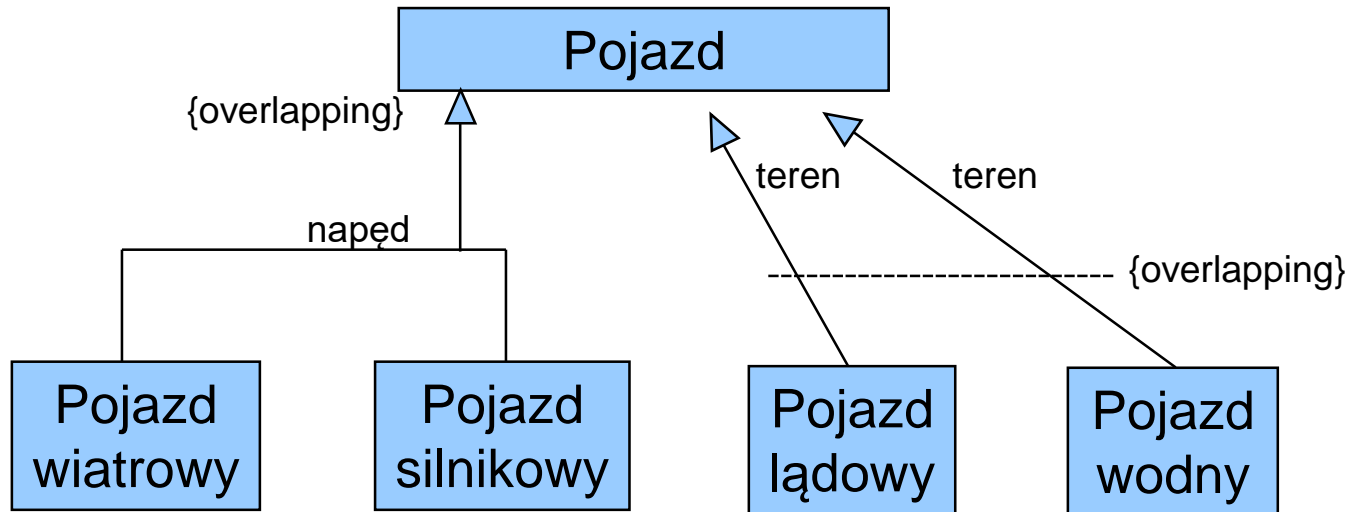
## Disjoint (domyślne)

- Podklasy wzajemnie się wykluczają (nie można wykorzystać wielodziedziczenia)

## Overlapping

- Podklasy nie wykluczają się wzajemnie (wielodziedziczenie jest możliwe)

# Ograniczenia dziedziczenia - przykłady

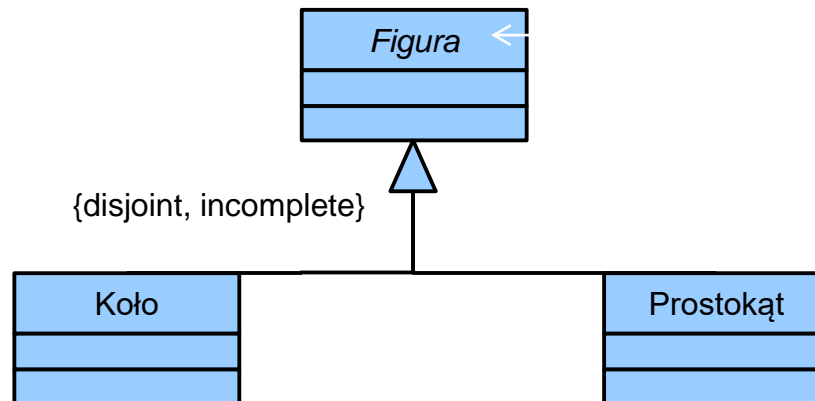


aspekt specjalizacji  
(dyskryminator)

# Klasa abstrakcyjna i klasa konkretna

Klasa abstrakcyjna nie ma (nie może mieć) bezpośrednich wystąpień i służy wyłącznie jako nadklasa dla innych klas. Stanowi jakby wspólną część definicji grupy klas o podobnej semantyce.

Klasa konkretna może mieć (ma prawo mieć) wystąpienia bezpośrednie



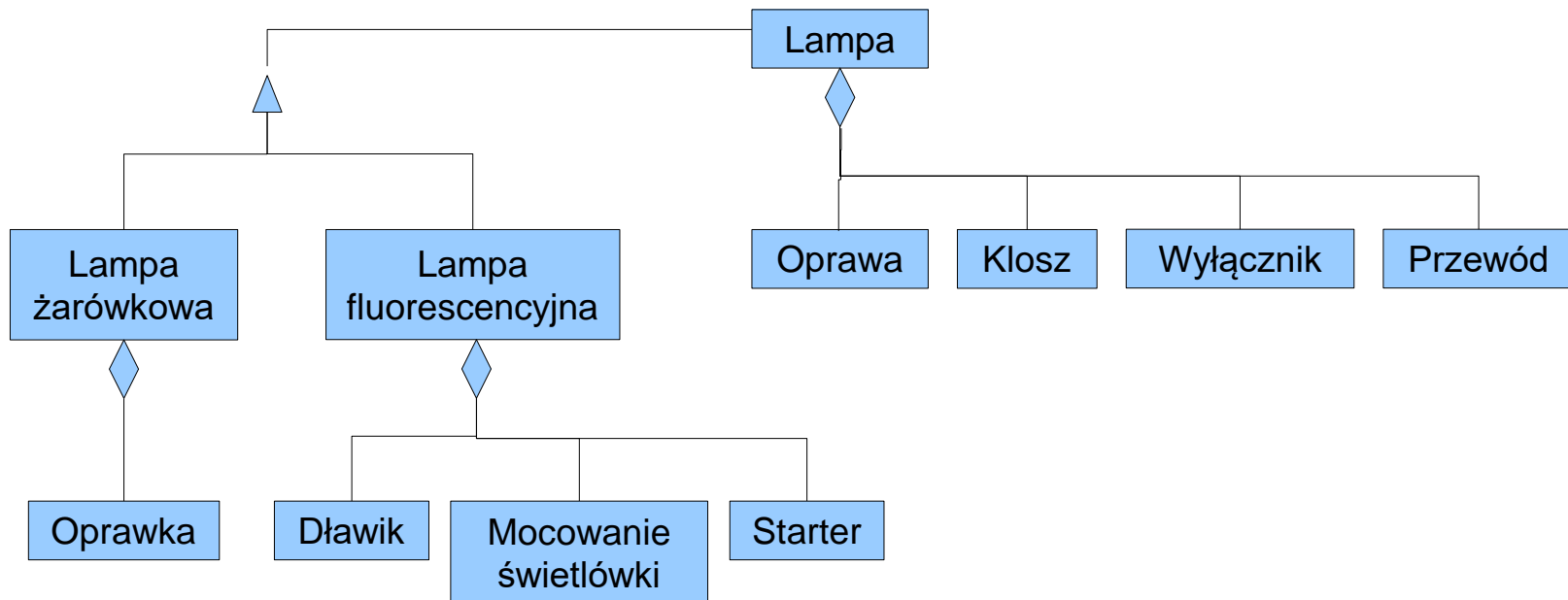
Klasa abstrakcyjna  
(nazwa pisana kursywą)

Klasy konkretne



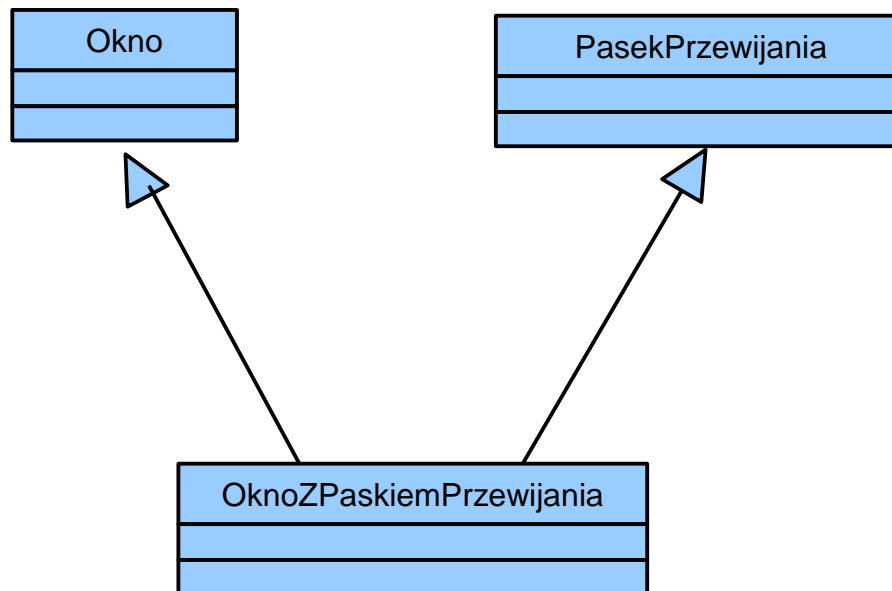
# Agregacja vs. generalizacja

- Wystąpienie agregacji wiąże dwa obiekty.
- Nie można mówić o wystąpieniu generalizacji; jest to związek między klasami.

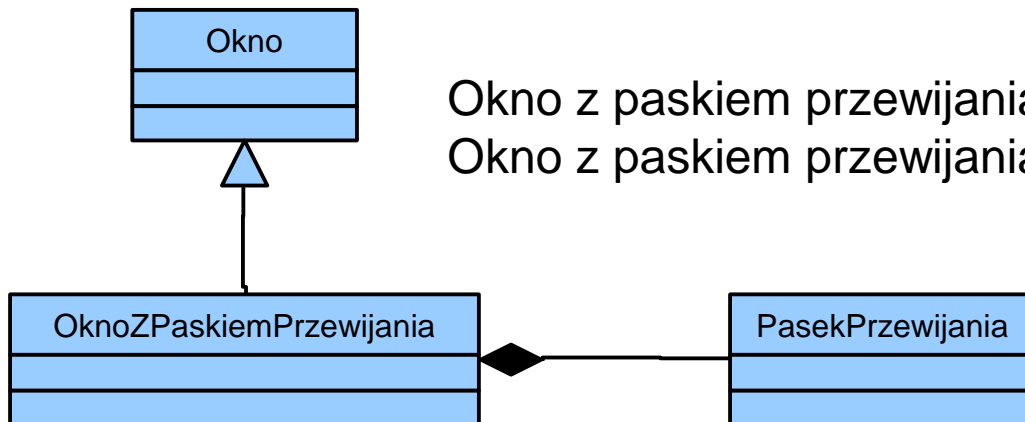
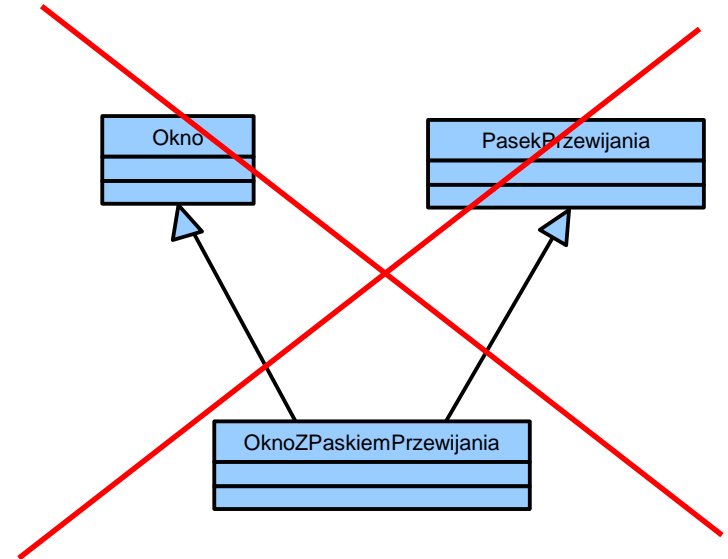


# Generalizacja a agregacja

- Generalizacja opisuje związek „jest” lub „jest typu”
- Agregacja reprezentuje związek „jest częścią”



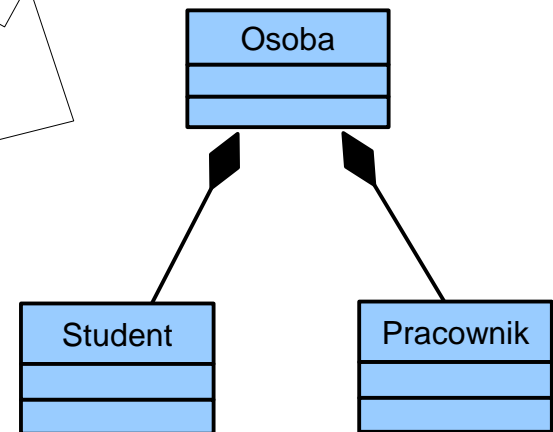
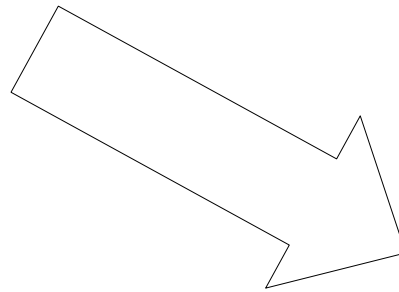
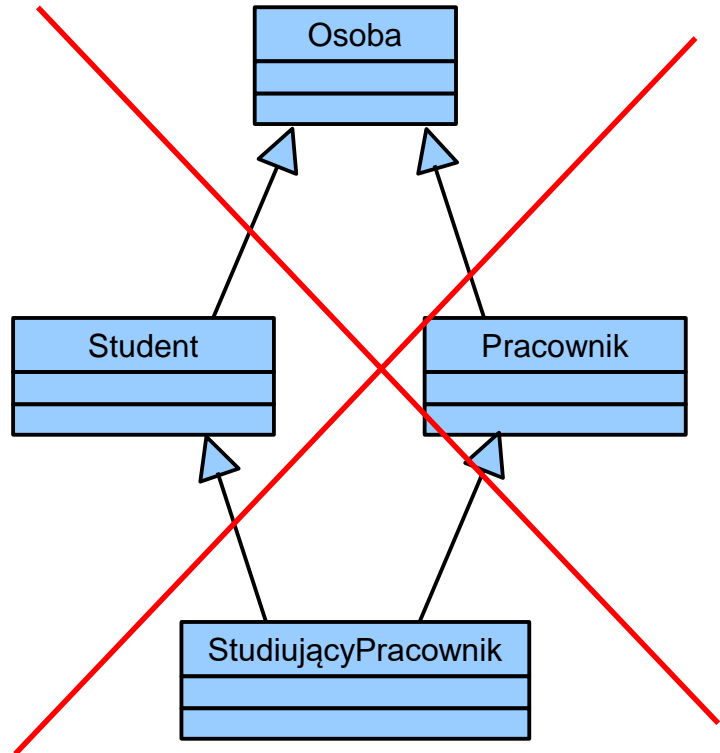
# Generalizacja a agregacja



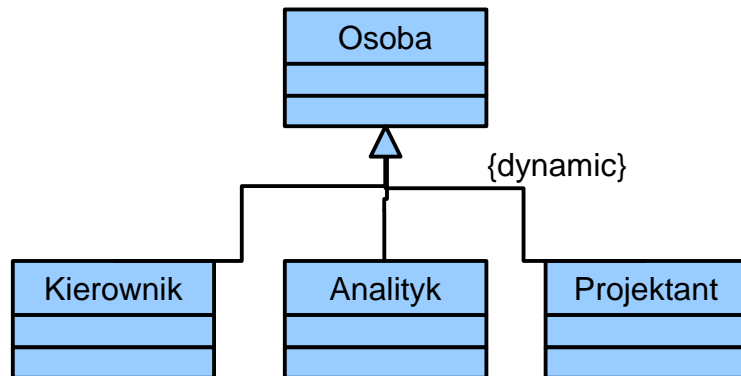
Okno z paskiem przewijania jest oknem

Okno z paskiem przewijania posiada pasek przewijania

# Obejście dziedziczenia wielokrotnego



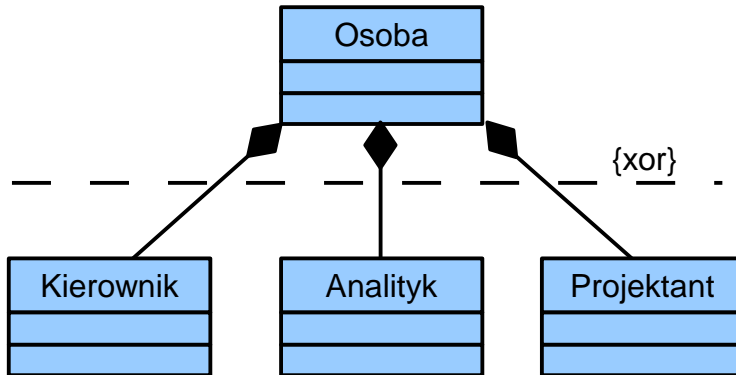
# Dziedziczenie dynamiczne



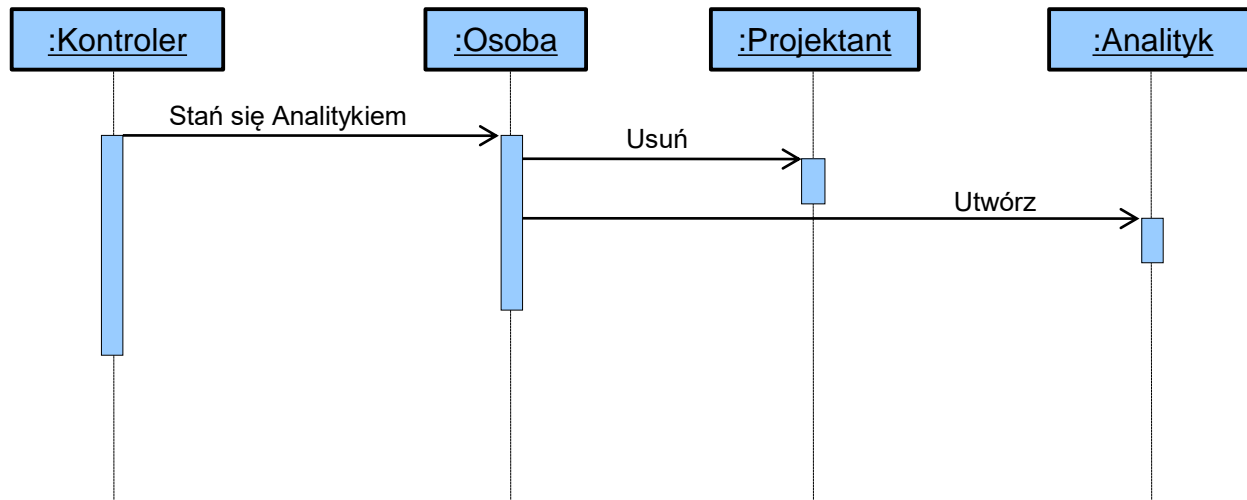
Osoba może zmieniać stanowisko, co można modelować poprzez tzw. Dziedziczenie dynamiczne. Przydatne dla modelowania koncepcyjnego, ale może być trudne w implementacji.

Przy zmianie specjalizacji Osoby powstaje nowy obiekt z którejś z trzech podklas: Kierownik, Analitik czy Projektant. Własności odziedziczone z klasy Osoba są każdorazowo przepisywane. Klasa osoba może być klasą abstrakcyjną

# Obejście dziedziczenia dynamicznego


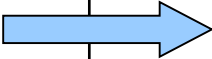


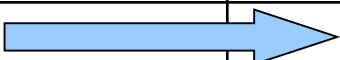
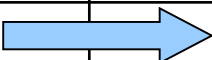


Usuwany jest obiekt związany ze starym stanowiskiem, tworzony jest obiekt przechowujący własności związane z nową specjalizacją oraz tworzone jest powiązanie między nowym obiektem a obiektem klasy **Osoba**, przechowującym dane osobowe. W tym przypadku klasa **Osoba** nie może być klasą abstrakcyjną.



# Projektowanie wymagań niefunkcjonalnych

W procesie analizy przypadków użycia przypisywaliśmy klasom mechanizmy analizy.

Mechanizm analizy	Mechanizm Projektowy	Mechanizm Implementacyjny
Trwałość	 Dane Spadkowe RDBMS	 ODBC
Trwałość	 Nowe dane OODBMS	 ObjectStore
Rozproszenie	 XML Web Services	 .Net Web Services (SOAP, WSDL)
<b>Analiza</b>	<b>Projektowanie</b>	<b>Implementacja</b>

# Projektowanie wymagań niefunkcjonalnych

- W tym kroku udoskonala się klasy projektowe, tak aby uwzględniały wymagania niefunkcjonalne.
- Należy tutaj wziąć pod uwagę specyfikację uzupełniającą oraz mechanizmy analizy określone w fazie analizy przypadków użycia.
- Teraz mechanizm analizy musi zostać przekształcony w odpowiadający, konkretny mechanizm projektowy.



# Projektowanie wymagań niefunkcjonalnych

Projektant powinien w tym kroku określić jak najwięcej właściwości mechanizmów projektowych. Może kierować się odpowiedziami na następujące pytania:

- Czy i jak wykorzystać istniejące produkty (np. komponenty)?
- Jak wymagania niefunkcjonalne zaadoptować do języka programowania i jego możliwości?
- Jak osiągnąć wymaganą wydajność?
- Jak osiągnąć wymagany poziom zabezpieczeń?
- Jak obsługiwać błędy?