

Dr Katarzyna Grzesiak-Kopeć

Inżynieria oprogramowania



4. Technologia obiektowa i język UML



Plan wykładu

- Tworzenie oprogramowania
- Najlepsze praktyki IO
- Inżynieria wymagań
- Technologia obiektowa i język UML
- Techniki IO
- Metodyki zwinne
- Refaktoryzacja
- Mierzenie oprogramowania
- Jakość oprogramowania
- Programowanie strukturalne
- Modelowanie analityczne
- Wprowadzenie do testowania

Technologia obiektowa

**Zestaw zasad tworzenia oprogramowania wraz
z językami, bazami danych i innymi
narzędziami, które te zasady wspomagają.**

Object Technology – A Manager's Guide, Taylor, 1997

Obiekty i najlepsze praktyki

- Rozwój iteracyjny
 - Toleruje zmiany w wymaganiach, integruje elementy progresywnie, ułatwia ponowne użycie
 - Architektura modułowa
 - Oczywiste
 - Wizualizacja modelu
 - Łatwo zrozumieć
 - Łatwo zmienić
- Zarządzanie wymaganiami
 - Ciągła weryfikacja jakości
 - Zarządzanie zmianami

Siła technologii

- Paradygmat jedności
 - Jeden język stosowany przez użytkowników, analityków, projektantów oraz programistów
- Ułatwia ponowne wykorzystanie architektury i kodu
- Model lepiej oddaje rzeczywistość
- Stabilność
- Adaptacja zmian

Technologia obiektowa

Abstrakcja

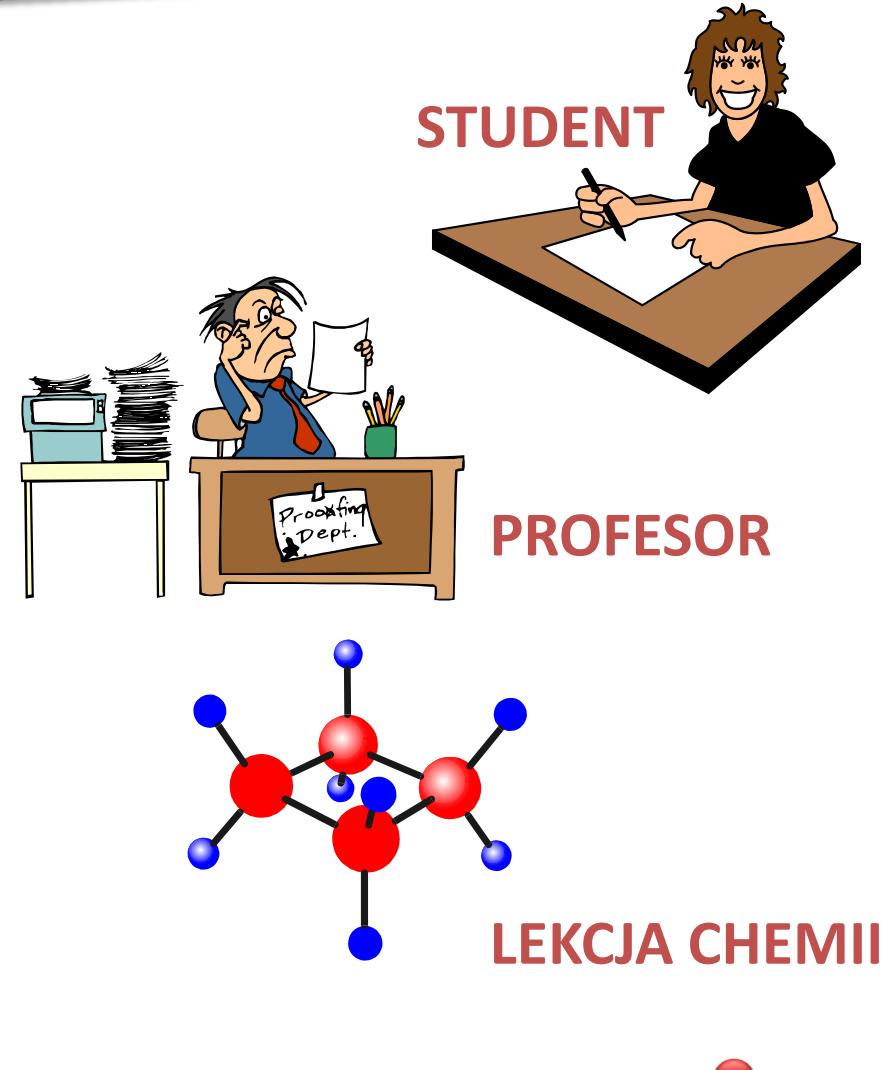
Hermetyzacja
(Enkapsulacja)

Modularność

Hierarchia

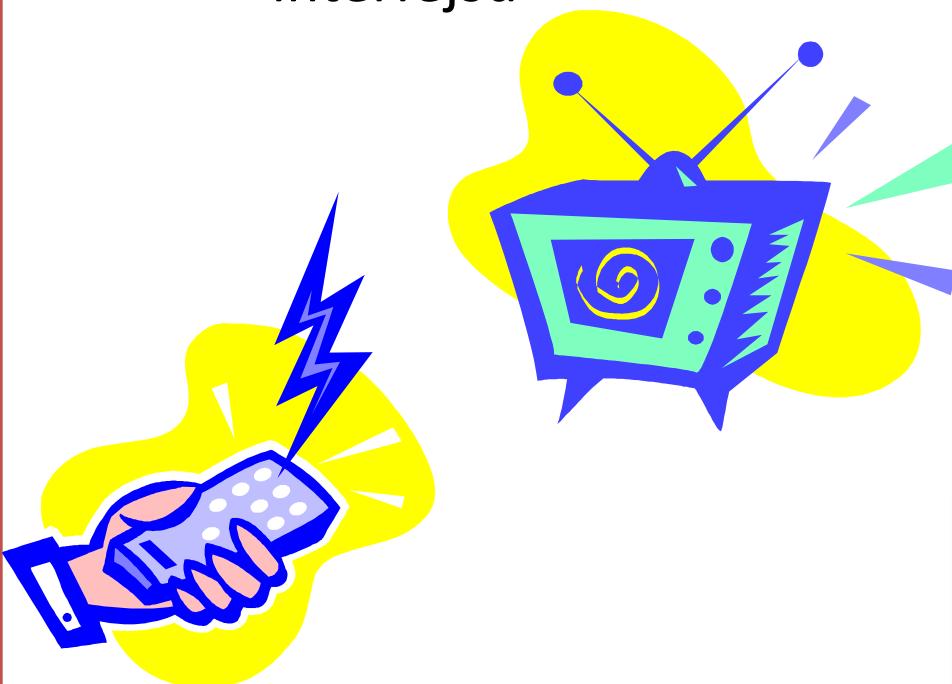
Abstrakcja

- Istotna charakterystyka jednostki odróżniająca ją od wszystkich jednostek innego typu
- Definiuje pewien ograniczony obszar zależny od perspektywy
- Nie jest konkretnym przykładem, a jedynie oznacza istotę czegoś



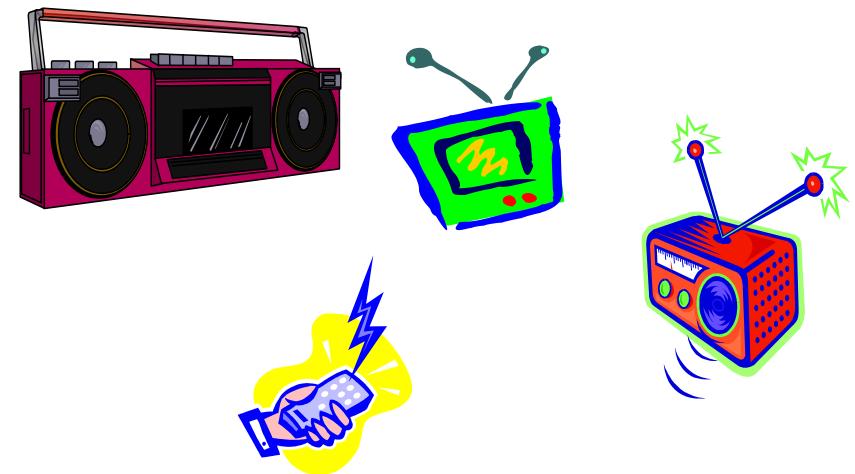
Hermetryzacja

- Ukrywanie przed klientem implementacji
 - Klient korzysta tylko z interfejsu



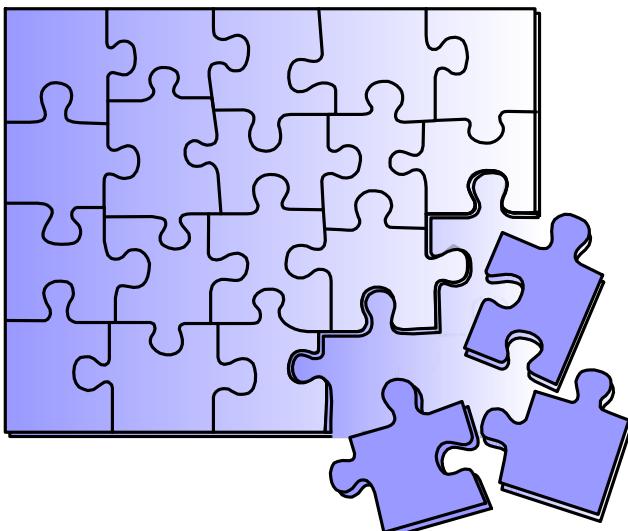
Polimorfizm

- Możliwość umieszczenia wielu różnych implementacji za jednym interfejsem



Modularność

- Podział całości na łatwe do zarządzania części
- Pomaga ogarnąć i zrozumieć skomplikowane systemy



Zapisy na kursy



Opłaty



Studenci



Katalog kursów

Hierarchia

Wzrasta poziom abstrakcji

Majątek

Konta
bankowe

Środki
trwałe

Zabezpieczenia

Walutowe

Złotówkowe

Akcje

Weksle

Maleje poziom abstrakcji

Co to jest OBIEKT?

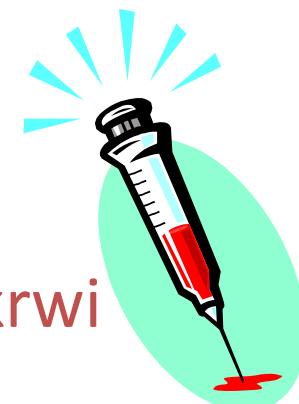
Reprezentuje pewną jednostkę albo fizyczną, albo koncepcyjną, albo oprogramowania.

✓ Jednostka fizyczna



✓ Jednostka konceptualna

proces pobierania krwi



✓ Jednostka oprogramowania



Obiekt – formalna definicja

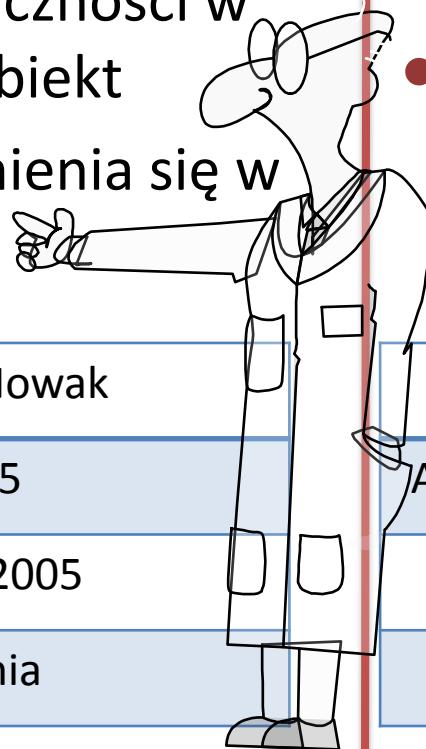
Obiekt jest jednostką o dobrze zdefiniowanym zakresie, charakteryzującą się pewnym **stanem i zachowaniem**.

- Stan jest reprezentowany przez:
 - atrybuty i relacje
- Zachowanie jest reprezentowane przez:
 - operacje, metody i automaty stanów

Obiekt

Jest w jakimś stanie

- Stan obiektu jest to jedna z możliwych okoliczności w jakich istnieje obiekt
- Stan obiektu zmienia się w czasie



Ma jakieś zachowanie

- Określa jak obiekt się zachowuje i jak reaguje
- Widoczne zachowanie jest modelowane przez wiadomości, na które umie odpowiedzieć

Imię i nazwisko:	Jan Nowak	Wystawia stopnie:	wystawStopnie()
Identyfikator:	12345	Akceptuje plan zajęć:	akceptujPlanZajec()
Data zatrudnienia:	1.IX.2005	Bierze urlop:	wezUrlop()
Przedmiot:	chemia	Wpisuje uwagi:	dodajUwage()

Obiekt jest unikalny

Każdy obiekt jest **unikalną** jednostką nawet, jeżeli jest w takim samym stanie jak inny obiekt.



Co to jest klasa?

- Klasa jest opisem zbioru obiektów posiadających ten sam zbiór atrybutów, operacji, relacji oraz semantykę
 - Obiekt jest instancją klasy
- Klasa jest abstrakcją, a więc:
 - Podkreśla pewną charakterystykę
 - Ukrywa inne cechy

UML 2

- 1999 UML 2.0
- ...
- 2011 UML 2.4
- 2012 - 2013 UML 2.5

Diagramy UML 2.2

Diagramy struktur

1. Klas (Class Diag.)
2. Obiektów (Object Diag.)
3. Komponentów (Component Diag.)

UML 2.0 Mrożenia (Deployment Diag.)

5. Struktur złożonych (Composite Structure Diag.)
6. Pakietów (Package Diag.)
7. Profili (Profile Diag.)

UML 2.2

Diagramy zachowań

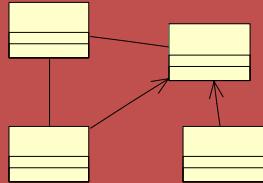
1. Przypadków użycia (Use Case Diag.)
2. Czynności (Activity Diag.)
3. Maszyny stanów (State Machine Diag.)
4. Komunikacji (Communication Diag.)

UML 2.0 Wencji (Sequence Diag.)

6. Interakcji czasowych (Timing Diag.)
7. Przeglądu interakcji (Interaction Overview Diag.)

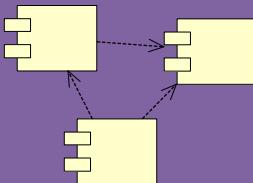


Architektura: 4+1 Perspektywy



Logiczna

Struktura
Analityk, Projektant

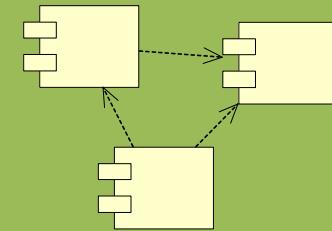


Procesy i procesory
Programista, Integrator

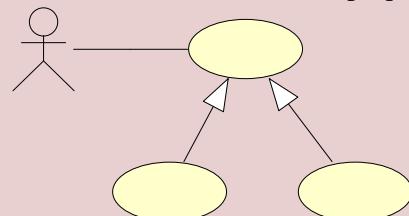
Procesowa

Implementacyjna

Moduły, interfejsy, zależności
Programista

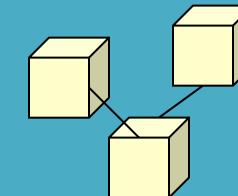


Przypadków
użycia



Funcjonalność
Użytkownik

Infrastruktura
Integrator, instalator



Wdrożenia

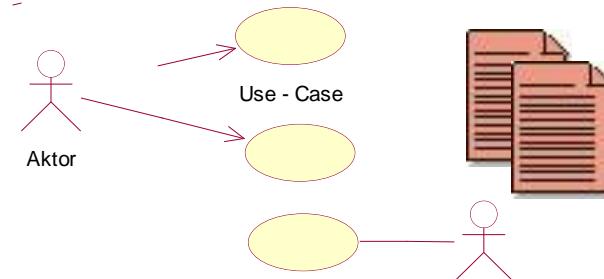
1. Zbieramy wymagania

Cel etapu

- Sprecyzowanie i uzgodnienie z klientem co system ma robić
- Przekazanie **właściwych** informacji programistom
- Ograniczenie systemu
- Baza dla planu pracy
- Baza dla oszacowania kosztów
- Definicja interfejsu systemu

Wynik etapu

- Model przypadków użycia (PU)
 - Diagramy PU
 - Specyfikacje PU
- Specyfikacje wymagań



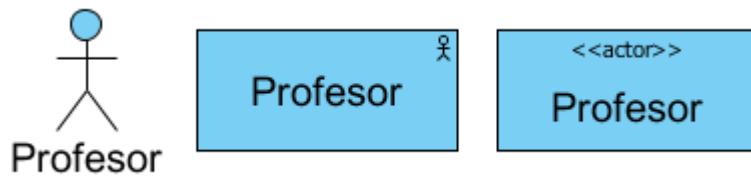
Zachowanie systemu

- Określa jak system się zachowuje i jak reaguje
- Jest widoczne na zewnątrz i możliwe do przetestowania
- Jest opisane przez PRZYPADKI UŻYCIA (Use-Cases)
- Przypadki użycia opisują system, jego środowisko i relacje pomiędzy systemem i środowiskiem

Główne pojęcia

Aktor

- Ktoś/coś kto/co wchodzi w interakcję z systemem z zewnątrz
 - Inicjuje akcje
 - Może reprezentować osobą fizyczną, rolę w systemie lub systemem zewnętrzny



Model-Code-Deploy Platform

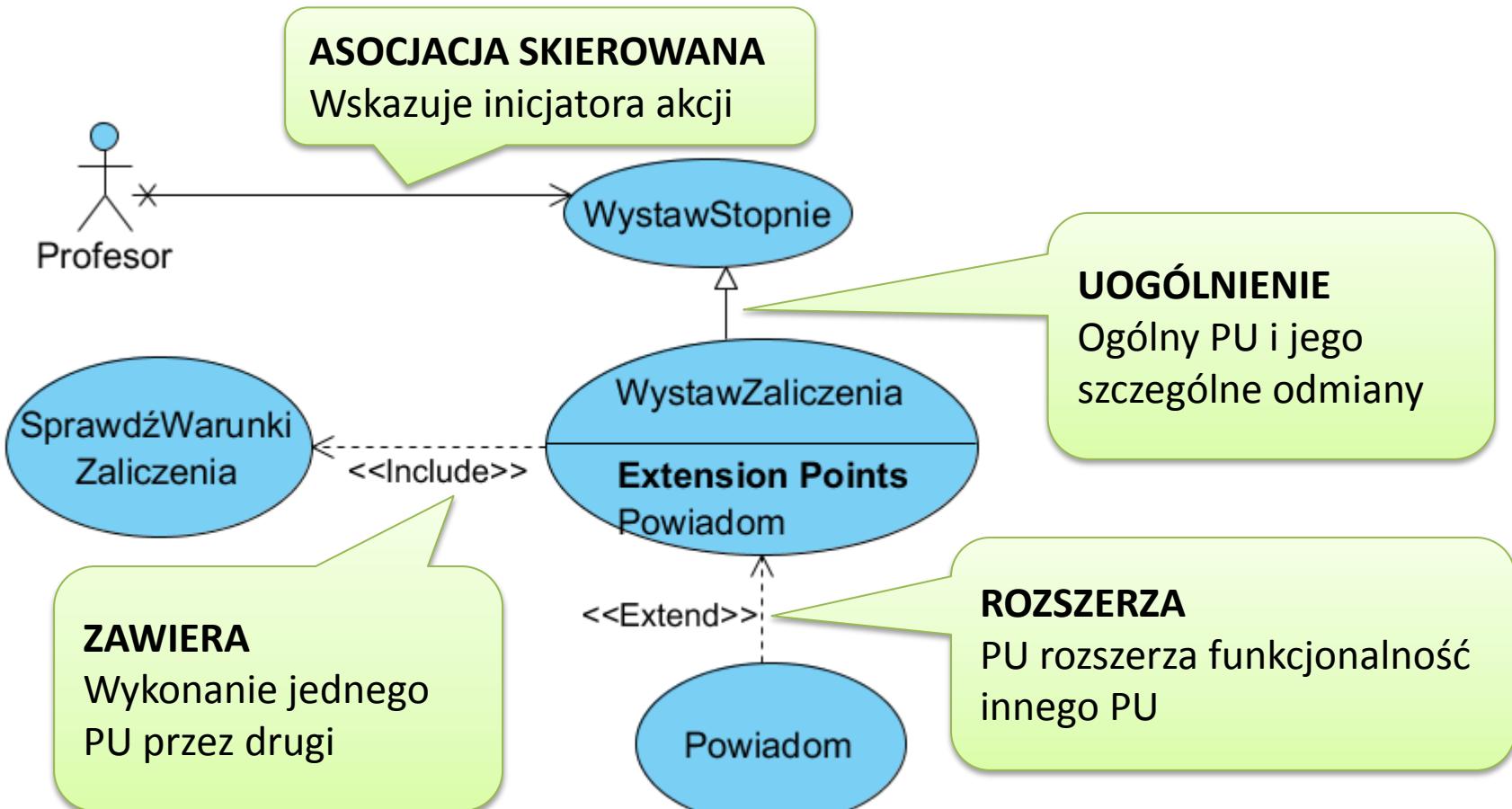
Przypadek użycia

- Sekwencja akcji wykonywanych przez system, których wynikiem jest konkretny (obserwowalny) rezultat mający znaczenie dla konkretnego aktora



Diagram Przypadków Użycia

- Opisuje wymagania funkcjonalne



Jak przeczytać diagram?

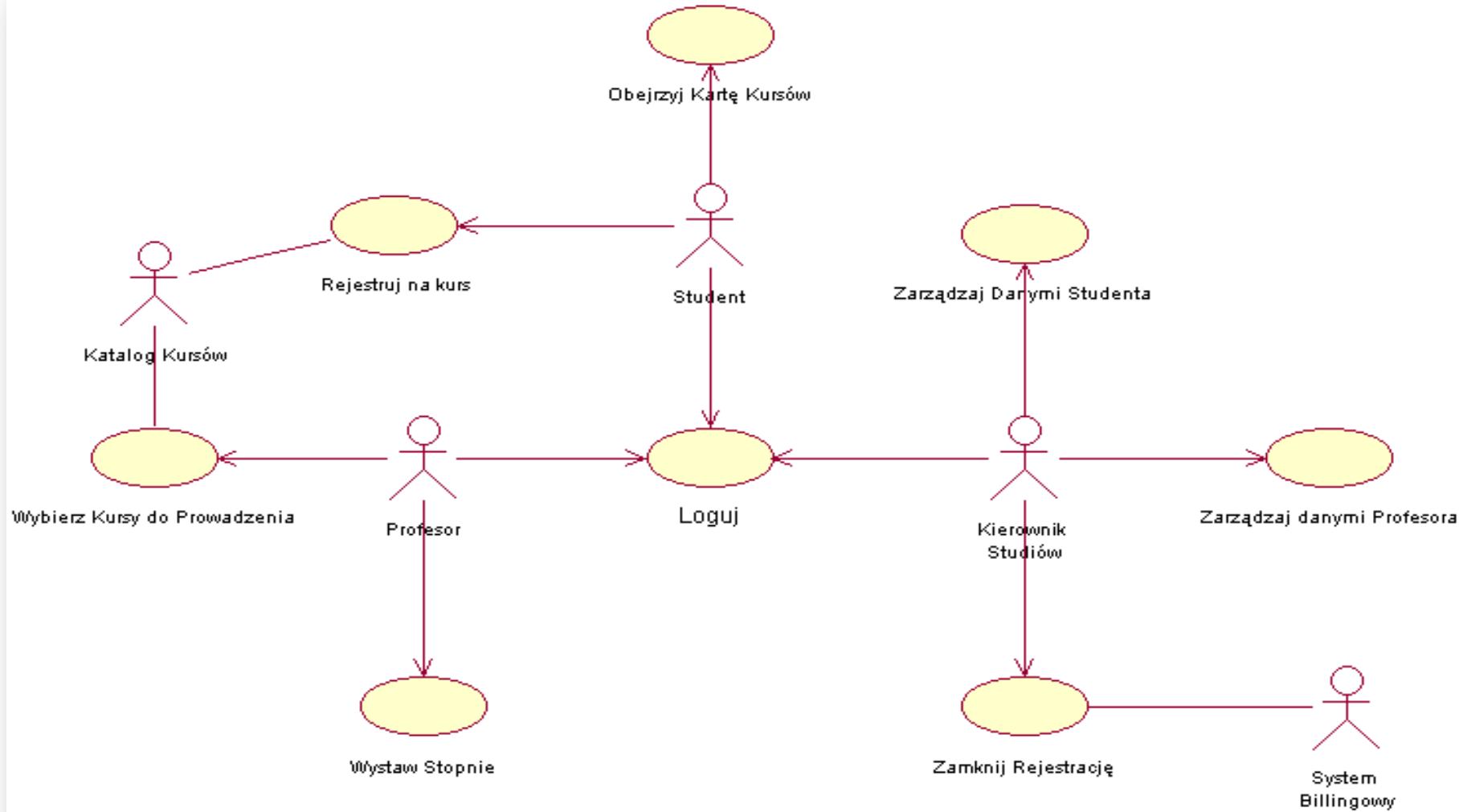
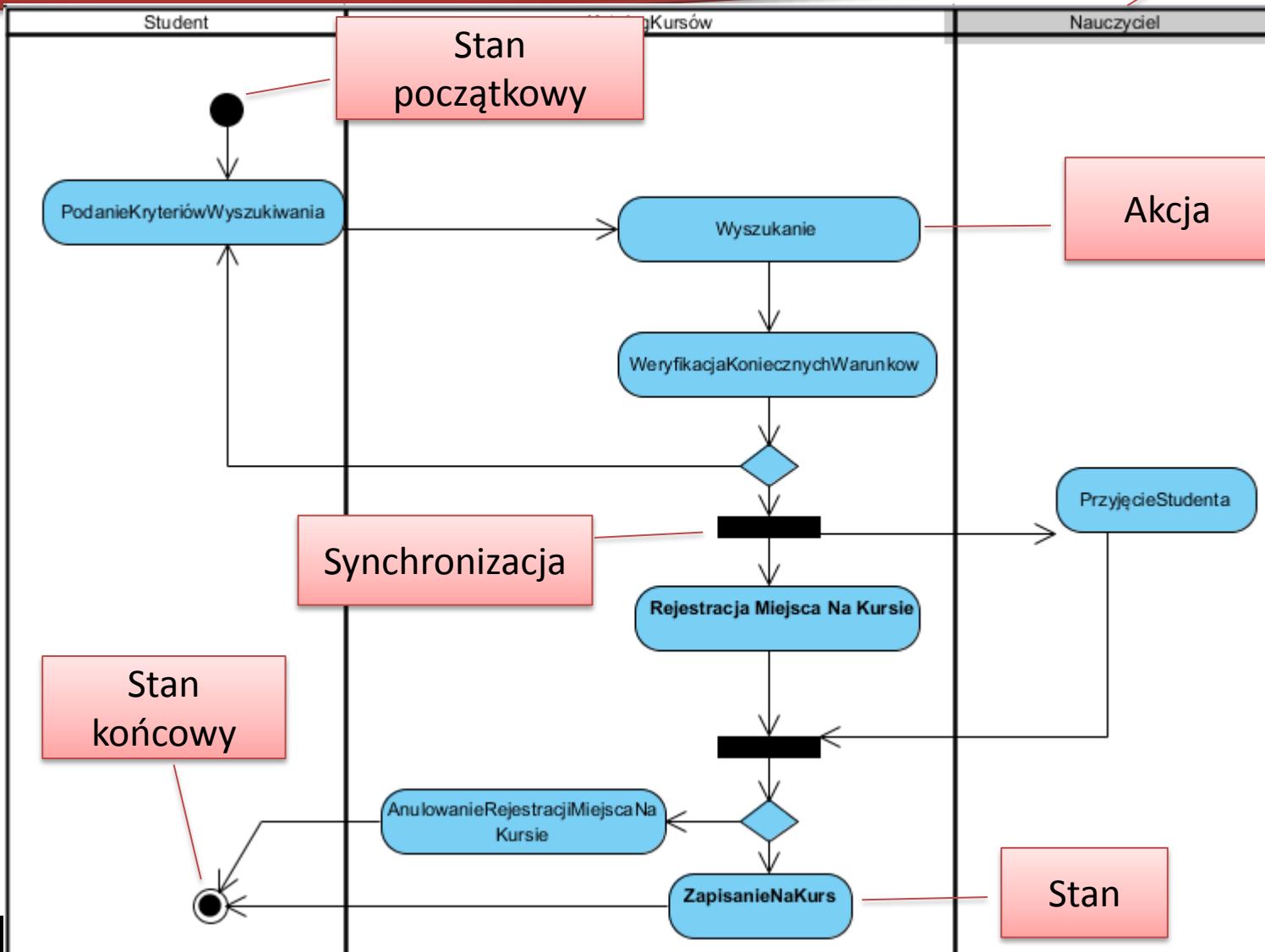


Diagram czynności

- Reprezentuje przebieg zdarzeń od akcji do akcji oraz wynikające z nich zmiany stanów
 - Automat skończenie stanowy, deterministyczny
- W modelu PU może być wykorzystany do przedstawienia kolejnych akcji w PU

Diagram czynności

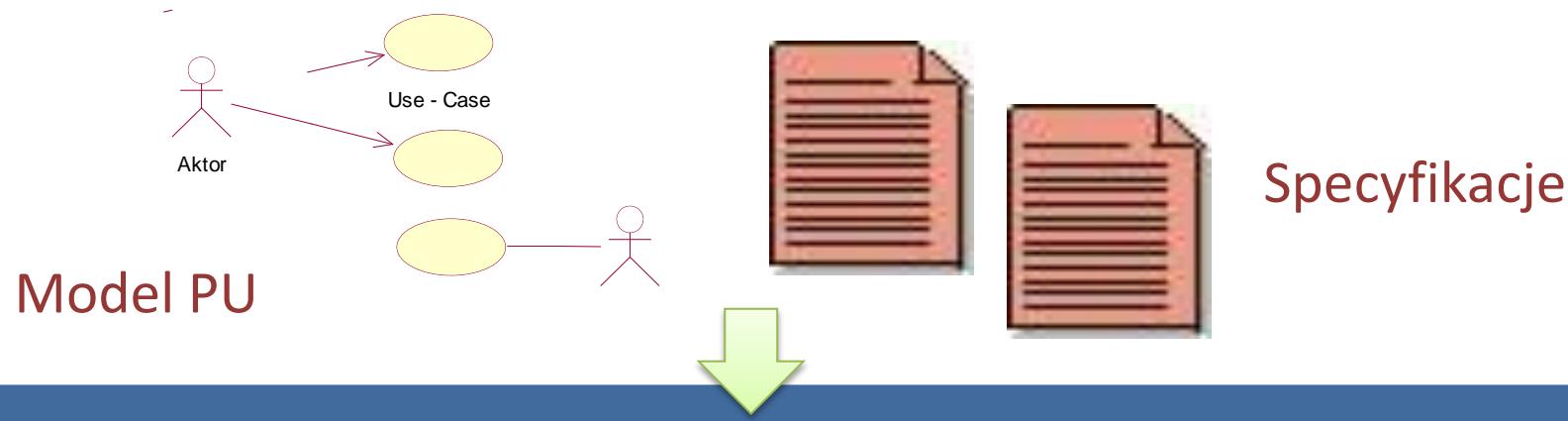
Tor



Zalety modelu

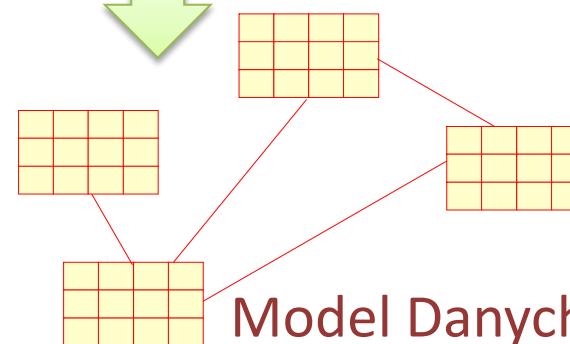
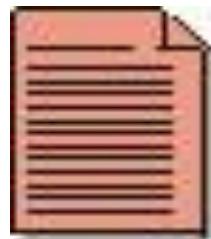
- Używany do komunikacji z klientami i ekspertami
 - Pozwala sprzedać produkt na wczesnym etapie rozwoju
 - Zapewnia jednolite zrozumienie wymagań
- Używany do identyfikacji:
 - Kto wchodzi w interakcję z systemem i jak ma reagować
 - Interfejsów jakie powinien posiadać system
- Używany do weryfikacji:
 - Czy zostały wychwycone wszystkie wymagania?
 - Czy zespół oprogramowania rozumie wymagania?

2. Analiza i projektowanie

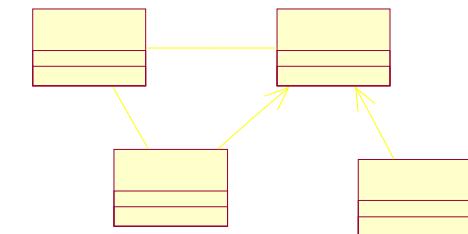


Analiza i projektowanie

Opis architektury
systemu



Model Danych

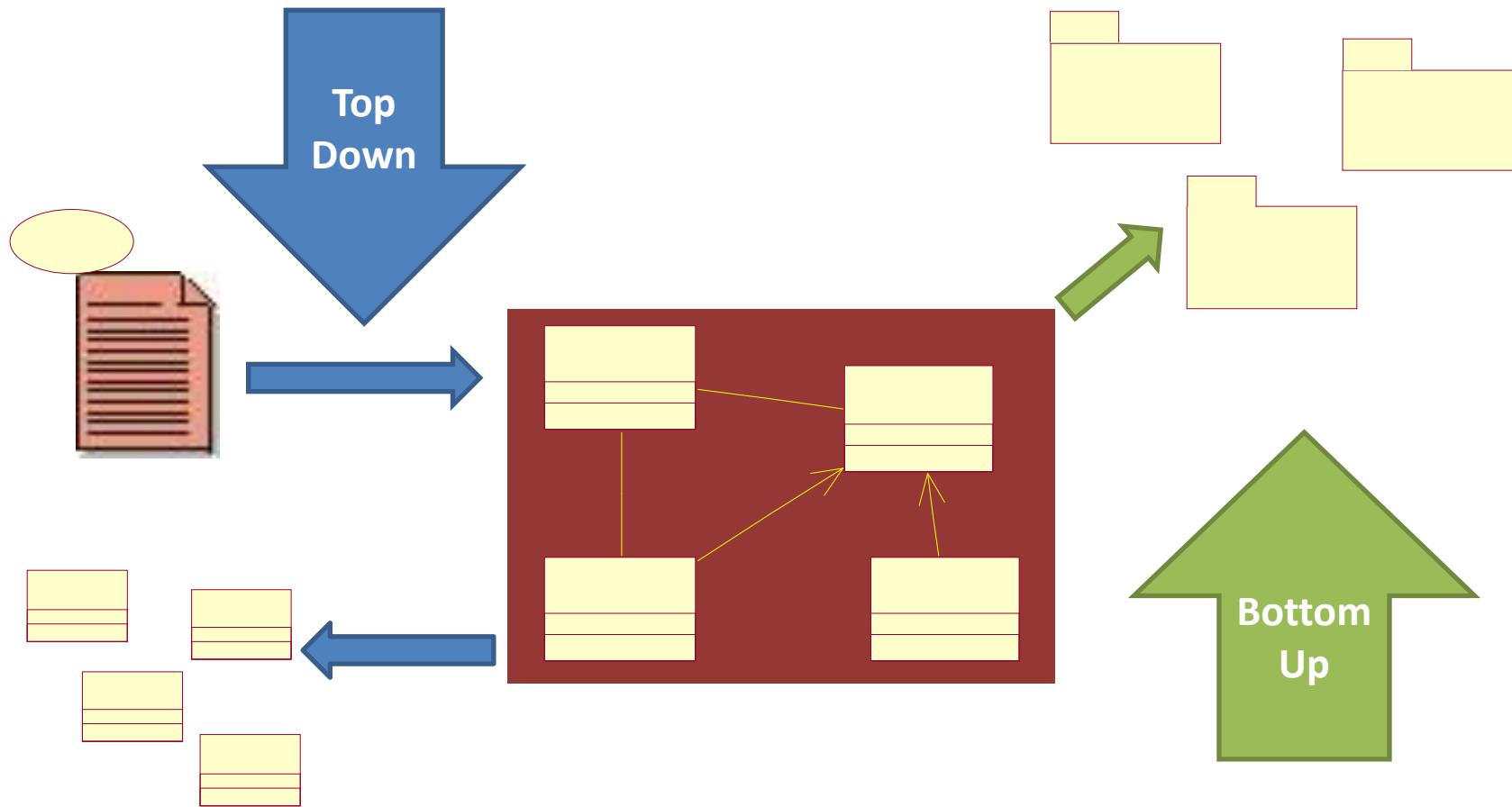


Projekt

Analiza vs Projektowanie

Analiza	Projektowanie
Zrozumieć problem	Rozwiązać problem
Ogólny projekt	Operacje i atrybuty
Zachowanie	Efektywność
Struktura systemu	Blisko kodowania
Wymagania funkcjonalne	Wymagania pozafunkcjonalne
Mały model ☺	Duży model

Ani Top-Down ani Bottom-Up

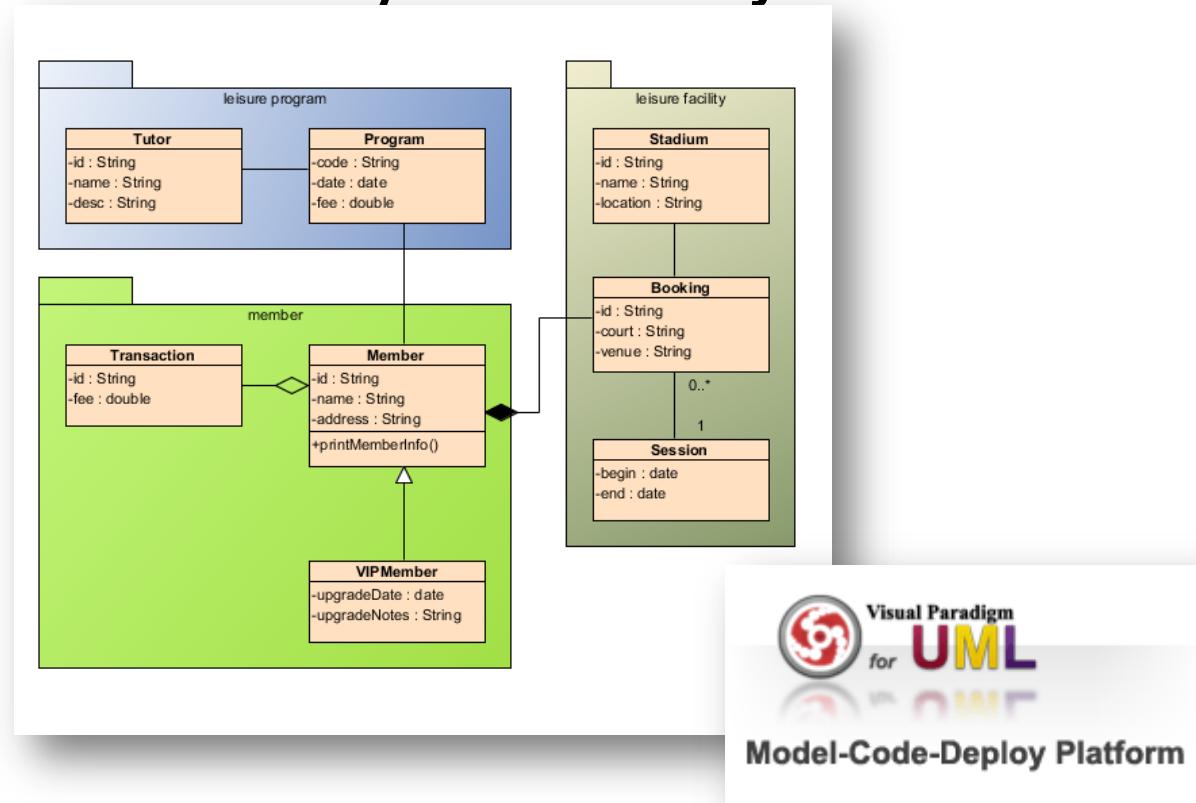


Wzorce i szkielety

- Wzorzec (pattern)
 - Znane rozwiązywanie znanego problemu w określonym kontekście
 - Fragment rozwiązania, element układanki
 - Wzorzec architektury: MVC
 - Wzorzec projektowy: fabryka abstrakcyjna
- Szkielet (framework)
 - Określa ogólne podejście do rozwiązywania problemu
 - Szkielet rozwiązania, którego elementami mogą być np. wzorce projektowe
 - Ruby on Rails, Spring Framework

Diagram klas

- Podstawowy diagram struktury logicznej
- Przedstawia klasy i ich relacje



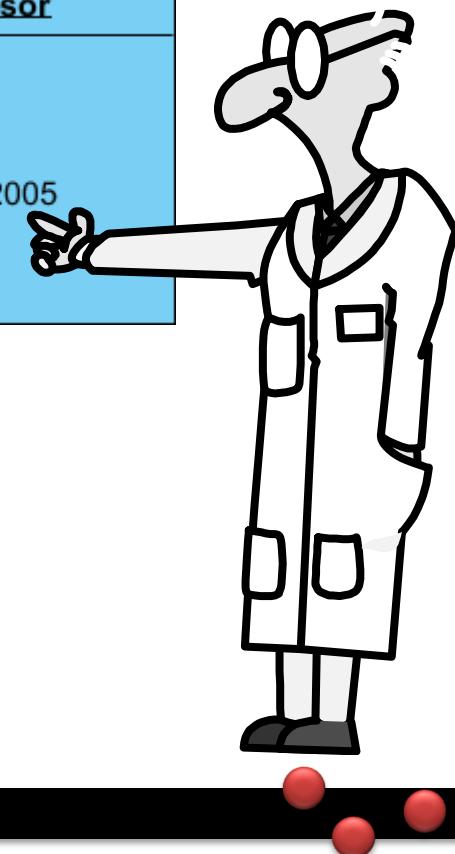
Klasy i obiekty w UMLu

Klasa

Profesor
-id : int = 0
-imie : String
-nazwisko : String
-dataZatrudnienia : Date
+KOD : String
+wystawStopnie(przedmiot : Przedmiot) : boolean
+wezUrlop(odDnia : Date, doDnia : Date) : boolean

Obiekt

Kowalski : Profesor
id = 12345
imie = Jan
nazwisko = Nowak
dataZatrudnienia = 1.IX.2005
KOD = AABBCC



Co to jest ATRYBUT?

- Atrybut jest nazwaną właściwością klasy, który opisuje zakres wartości jakie może przyjmować instancja właściwości
 - Klasa może mieć dowolną liczbę atrybutów lub nie mieć ich wcale

```
Profesor
-id : int = 0
-imie : String
-nazwisko : String
-dataZatrudnienia : Date
+KOD : String
+wystawStopnie(przedmiot : Przedmiot) : boolean
+wezUrlop(odDnia : Date, doDnia : Date) : boolean
```

Sygnatura atrybutu

widoczność

nazwa : typ

[krotność]

{ograniczenia}

= wartość domyślna

Profesor	
-	-id : int [1] {frozen} = 0
~	imie : String
-	nazwisko : String
-	dataZatrudnienia : Date
+	KOD : String
+	wystawStopnie(przedmiot : Przedmiot) : boolean
+	wezUrlop(odDnia : Date, doDnia : Date) : boolean

-

id : int

[1]

{frozen}

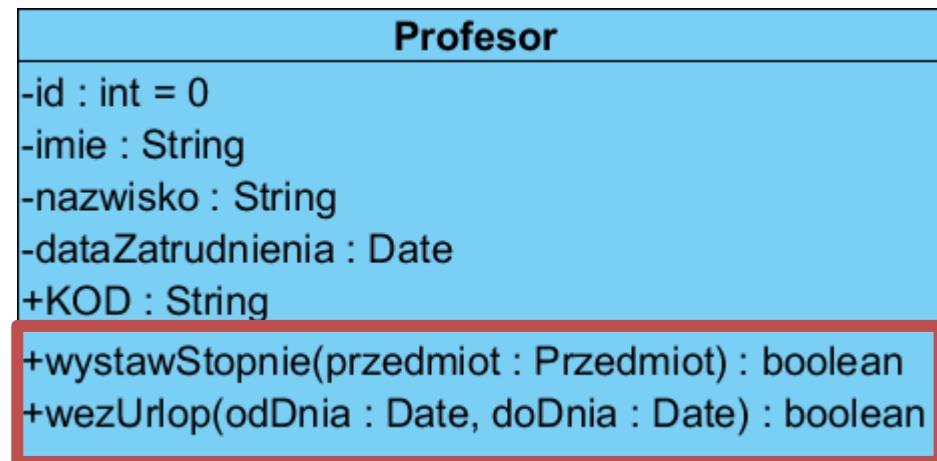
= 0

Sygnatura atrybutu

widoczność	nazwa : typ	[krotność]	{ograniczenia}	= wartość domyślna
+ publiczny	unspecified	1..*	{unique}	
- prywatny	0..1	*	{ordered}	
# chroniony	0..*	od..do	{nonunique}	
~ publiczny w pakiecie	1	2,4..6	{readOnly}	
-	id : int	[1]	{frozen}	= 0

Co to jest OPERACJA?

- Jest implementacją pewnej usługi, której wykonania można oczekiwac od każdego obiektu danej klasy
- Klasa może mieć dowolną liczbę operacji lub wcale



Sygnatura operacji

widoczność

nazwa

(parametr_1, .. parametr_n)

: typ

{ograniczenia}

Profesor

-id : int [1] {frozen} = 0
~imie : String
-nazwisko : String
-dataZatrudnienia : Date
+KOD : String

+wystawStopnie(in przedmiot : Przedmiot) : boolean
+wezUrlop(in odDnia : Date = 1.IX.2010, in doDnia : Date) : boolean

+

wezUrlop

(odDnia, doDnia)

: boolean

{}

Sygnatura operacji

widoczność

nazwa

(parametr_1, .. parametr_n)

: typ

{ograniczenia}

parametr_1

kierunek

nazwa : typ

[krotność]

= wartość domyślna

in

odDnia : Date

[]

= 1.IX.2010

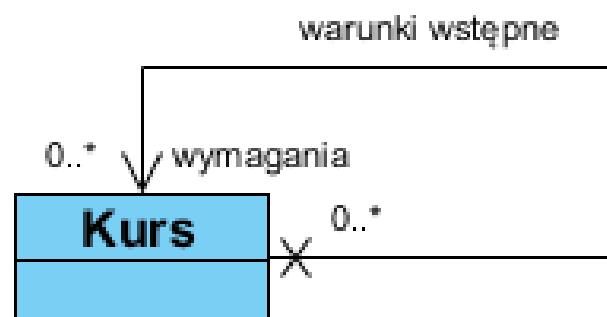
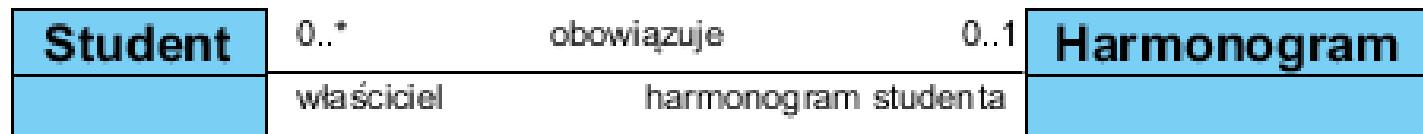
out

inout

return

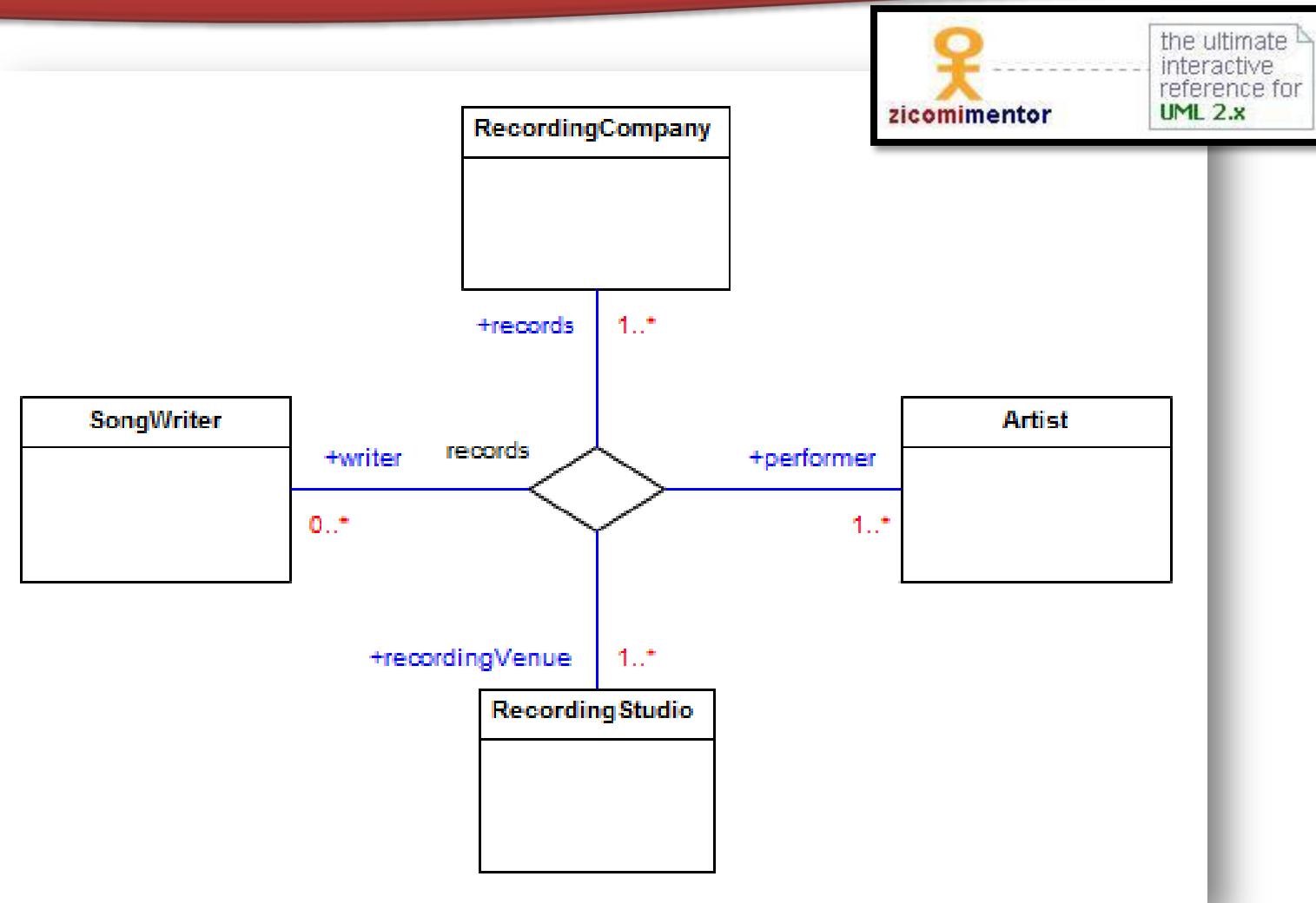
Asocjacja

- Semantyczna relacja pomiędzy dwoma lub wieloma klasyfikatorami, która reprezentuje zależność między ich instancjami
 - Strukturalna relacja określająca powiązanie



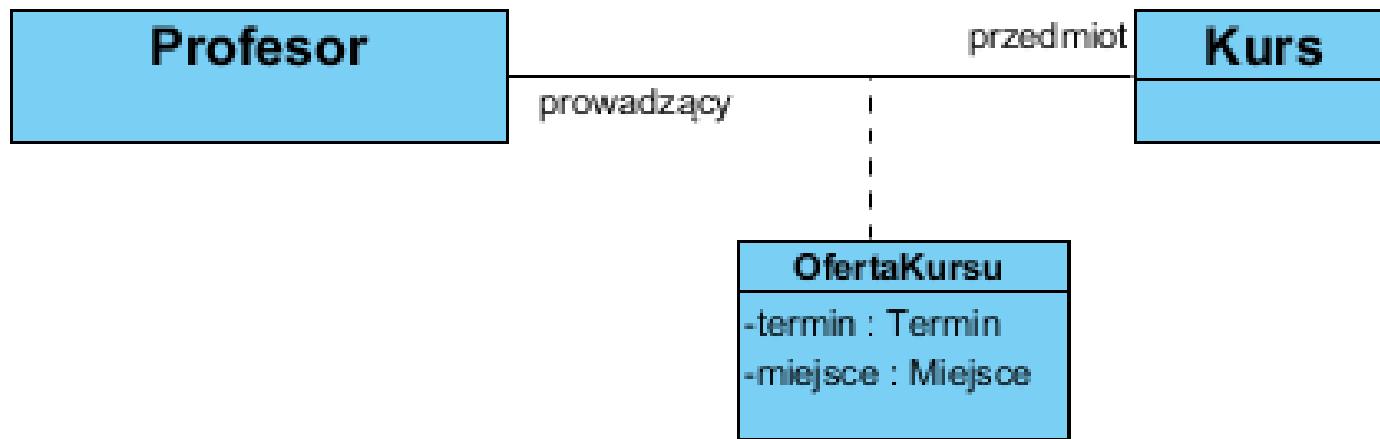
- Krotność
- Kierunek nawigacji
- Rola

Asocjacja n-arna



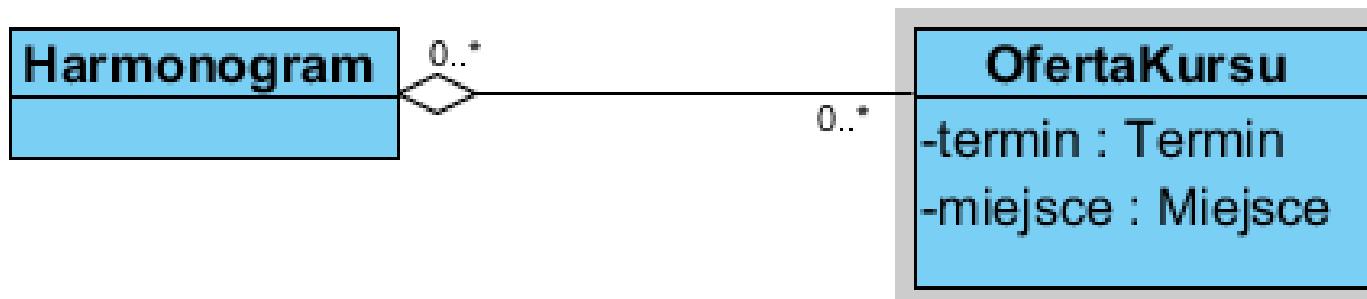
Klasa asocjacyjna

- Opisuje właściwości relacji asocjacji



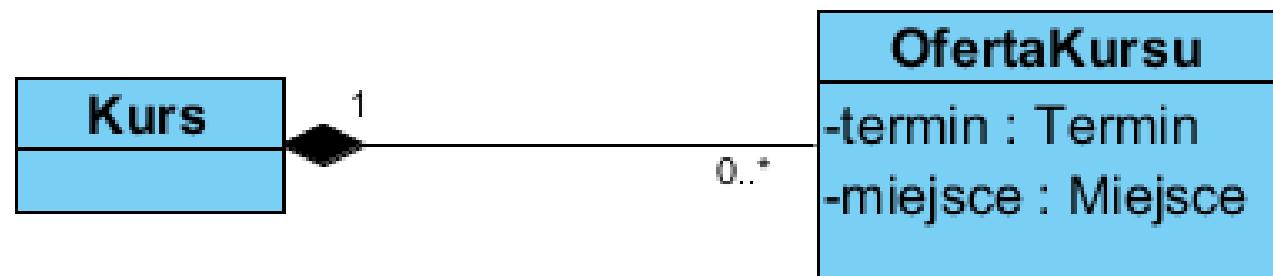
Agregacja

- Specjalny rodzaj asocjacji modelujący relację całość-część
- Część może należeć do kilku całości
- Całość nie zarządza czasem istnienia części



Kompozycja

- Specjalny rodzaj asocjacji modelujący relację całość-część
- Część może należeć do co najwyżej jednej całości
- Całość zarządza czasem istnienia części

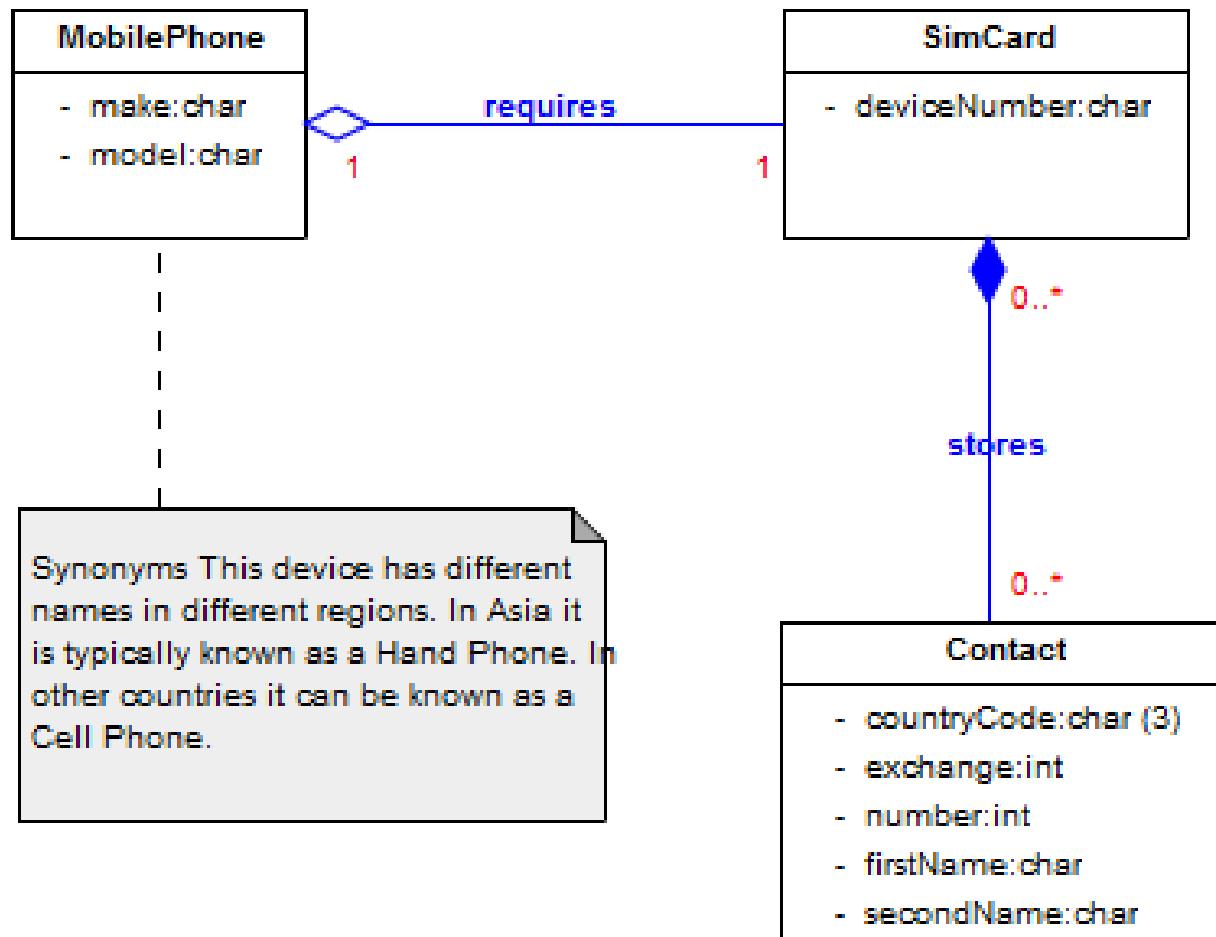


Agregacja i kompozycja



zicommentor

the ultimate
interactive
reference for
UML 2.x



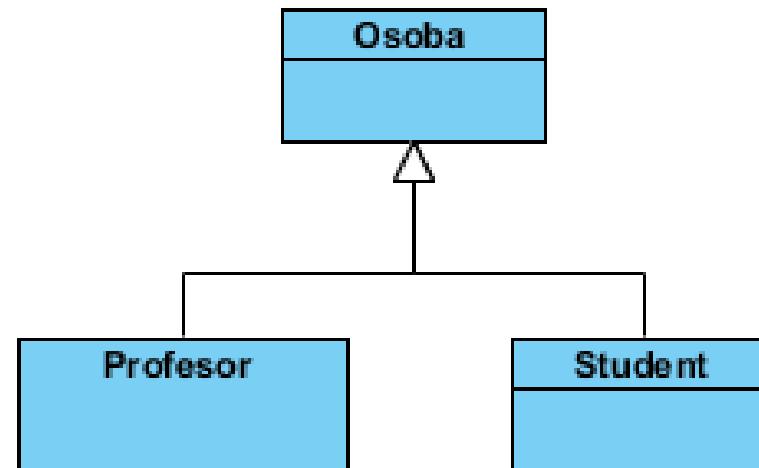
Zależność

- Jest to relacja pomiędzy dwoma elementami modelu, gdzie zmiana w jednym elemencie może wymusić zmianę drugiego
- Niestrukturalna relacja: **używa**
 - Występuje wtedy, gdy klasa jest parametrem lub zmienną lokalną metody innej klasy



Generalizacja

- Definiuje hierarchię od klas ogólnych do wyspecjalizowanych
- Techniki generalizacji
 - Dziedziczenie: **jest rodzajem**

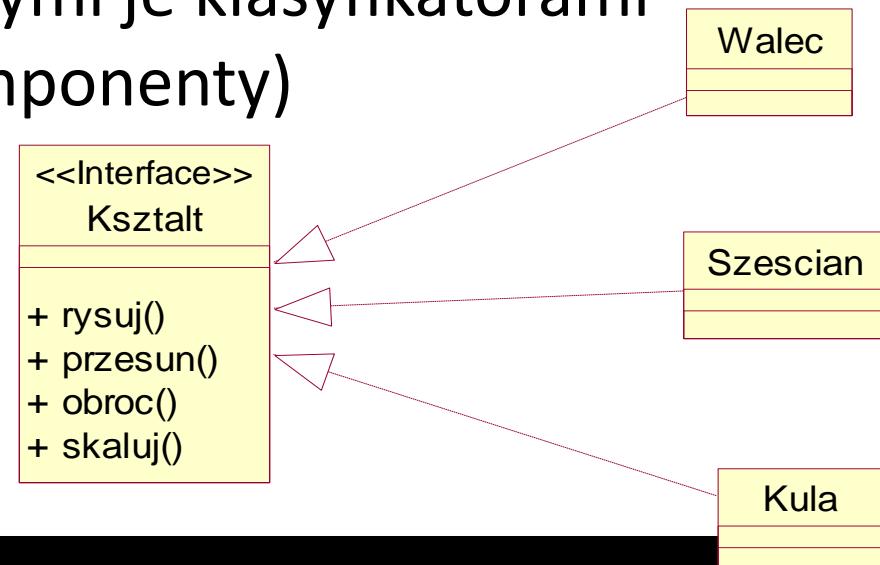


Co dziedziczymy?

- Podklasa dziedziczy od rodzica: atrybuty, operacje i relacje
- Podklasa może:
 - Dodać kolejne atrybuty, operacje i relacje
 - Przedefiniować dziedziczone operacje (ostrożnie)
- Wspólne atrybuty, operacje i/lub relacje są umieszczane na najwyższym możliwym poziomie abstrakcji

Realizacja

- Jeden klasyfikator jest zleceniodawcą dla drugiego (definiuje akcje, które drugi wykonuje)
- Może wystąpić między:
 - Interfejsami i realizującymi je klasyfikatorami (klasy, podsystemy, komponenty)
 - Przypadkami użycia



Klasa abstrakcyjna i interfejs

- Nie posiadają obiektów
- Klasa abstrakcyjna
 - Deklaruje wspólną funkcjonalność grupy klas
 - Może posiadać implementacje metod
- Interfejs
 - Deklaruje funkcjonalność bez implementacji
 - Czysto abstrakcyjny

Stereotyp

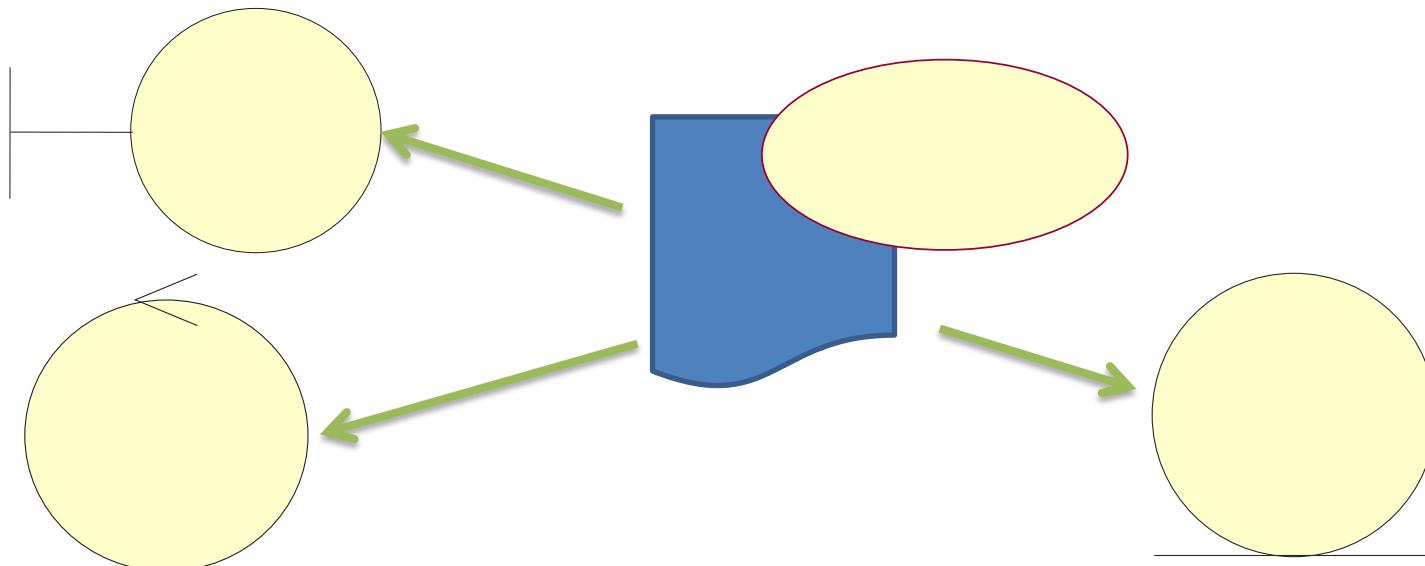
- Dodaje pewną semantykę do elementu

<<stereotyp>>
Klasa

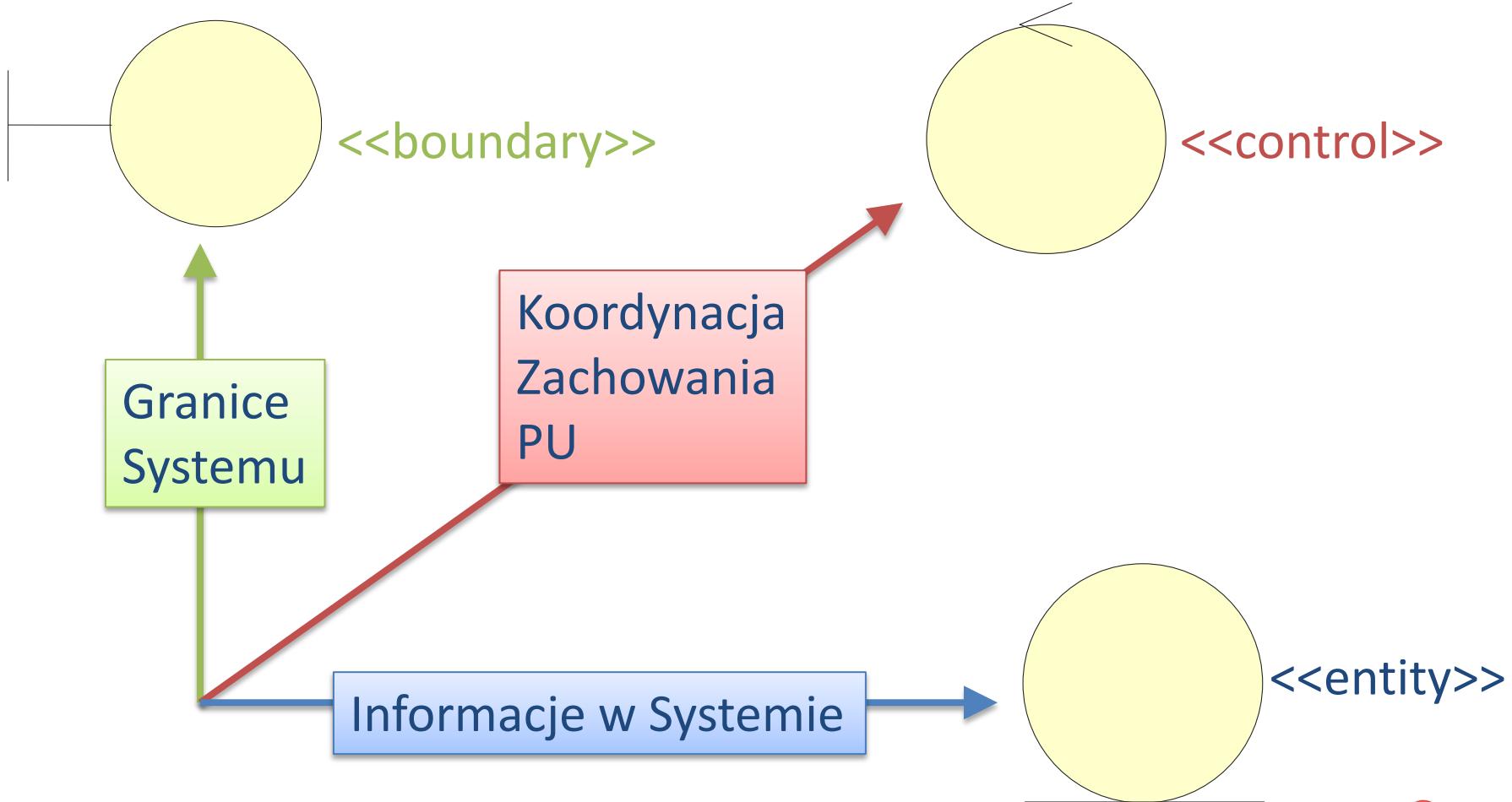
<<Interface>>
Interfejs

PU i klasy analityczne

- Chcemy przejść od modelu PU do projektu
- Kompletne zachowanie przypadku użycia musi być rozdysystrybuowane pomiędzy klasy analityczne

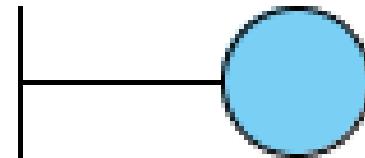


Klasy analityczne

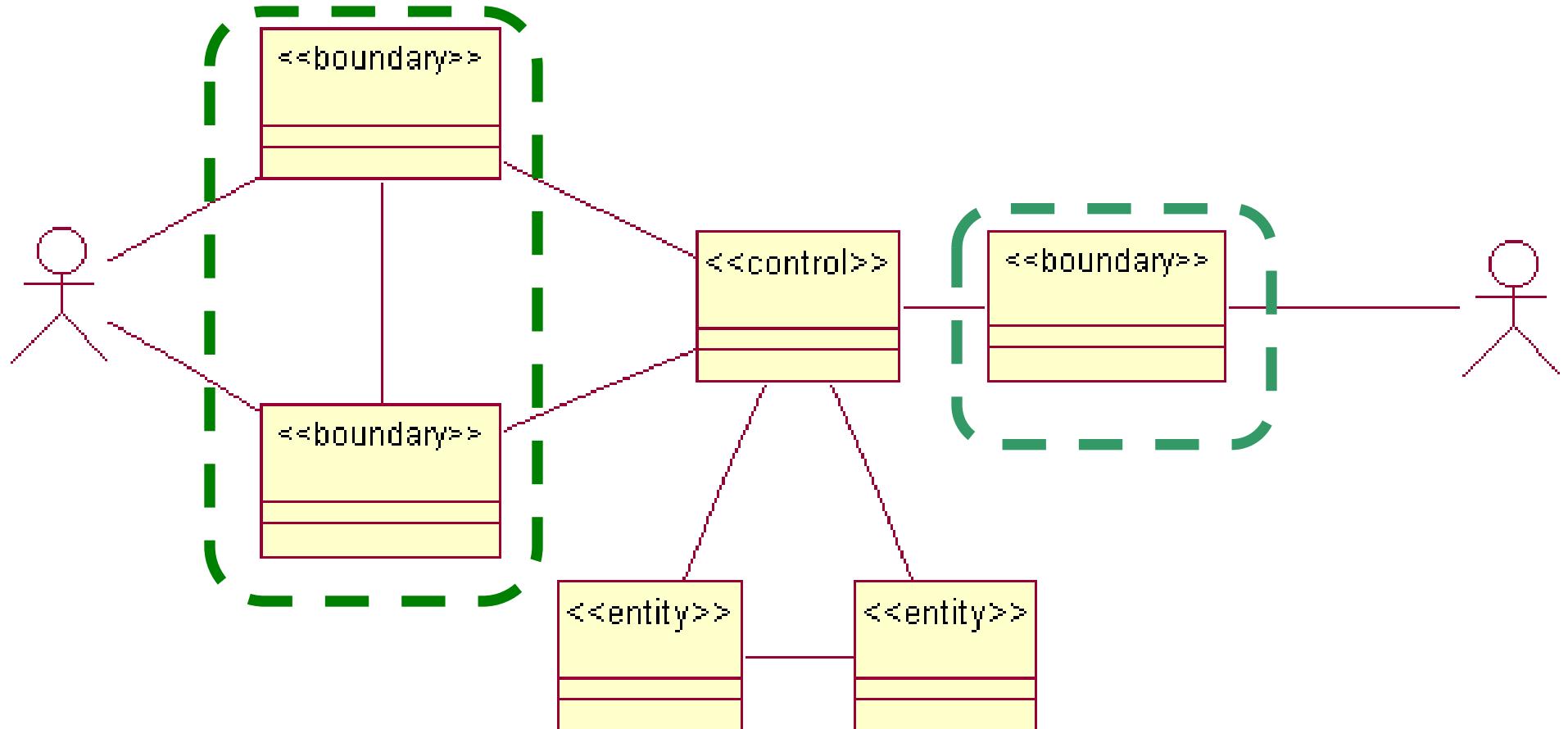


<<boundary>>

- Pośredniczy pomiędzy systemem i otoczeniem
- Rodzaje:
 - Klasy interfejsu użytkownika
 - Klasy interfejsu systemu
 - Klasy interfejsu urządzenia
- Co najmniej jedna klasa <<boundary>> na parę aktor-przypadek użycia



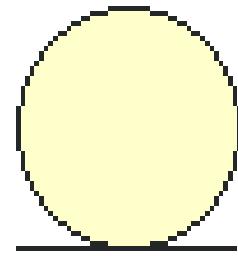
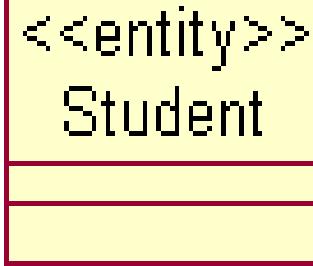
<<boundary>>



<<entity>>

- Reprezentuje przechowywane w systemie informacje
- Z reguły reprezentuje kluczowe pojęcia związane z systemem
- Szukamy ich w:
 - Słowniku
 - Scenariuszach opisujących przypadki użycia
 - Kluczowe abstrakcje

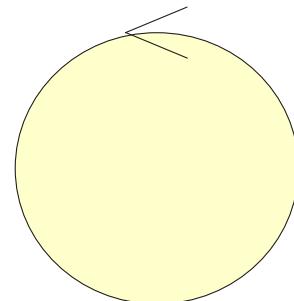
<<entity>>



OfertaKursu

<<control>>

- Koordynator zachowania.
- Jedna klasa typu <<control>> na jeden przypadek użycia



<<control>>

- Niezależna od otoczenia
 - Nie zmienia się jeśli ono się zmienia
- Definiuje logikę kontrolującą i transakcje pomiędzy PU
- Zmienia się w niewielkim stopniu jeśli zmieni się struktura lub zachowanie klas typu <<entity>>
- Używa lub ustala zawartość kilku klas typu <<entity>>, czyli koordynuje ich zachowanie
- Nie zachowuje się tak samo przy każdej aktywacji

Zachowanie klas

- Dla każdego scenariusza:
 - Identyfikujemy klasy
 - Rozdzielamy obowiązki pomiędzy klasy
 - Modelujemy interakcję tych klas
 - Diagram sekwencji
 - Diagram komunikacji
 - Diagram przeglądu interakcji
 - Diagram uwarunkowań czasowych

Jak rozdzielać obowiązki

- <<boundary>>
 - Komunikacja z aktorem
- <<entity>>
 - Zachowanie korzystające z hermetyzowanych przez klasę danych
- <<control>>
 - Zachowanie specyficzne dla danego scenariusza

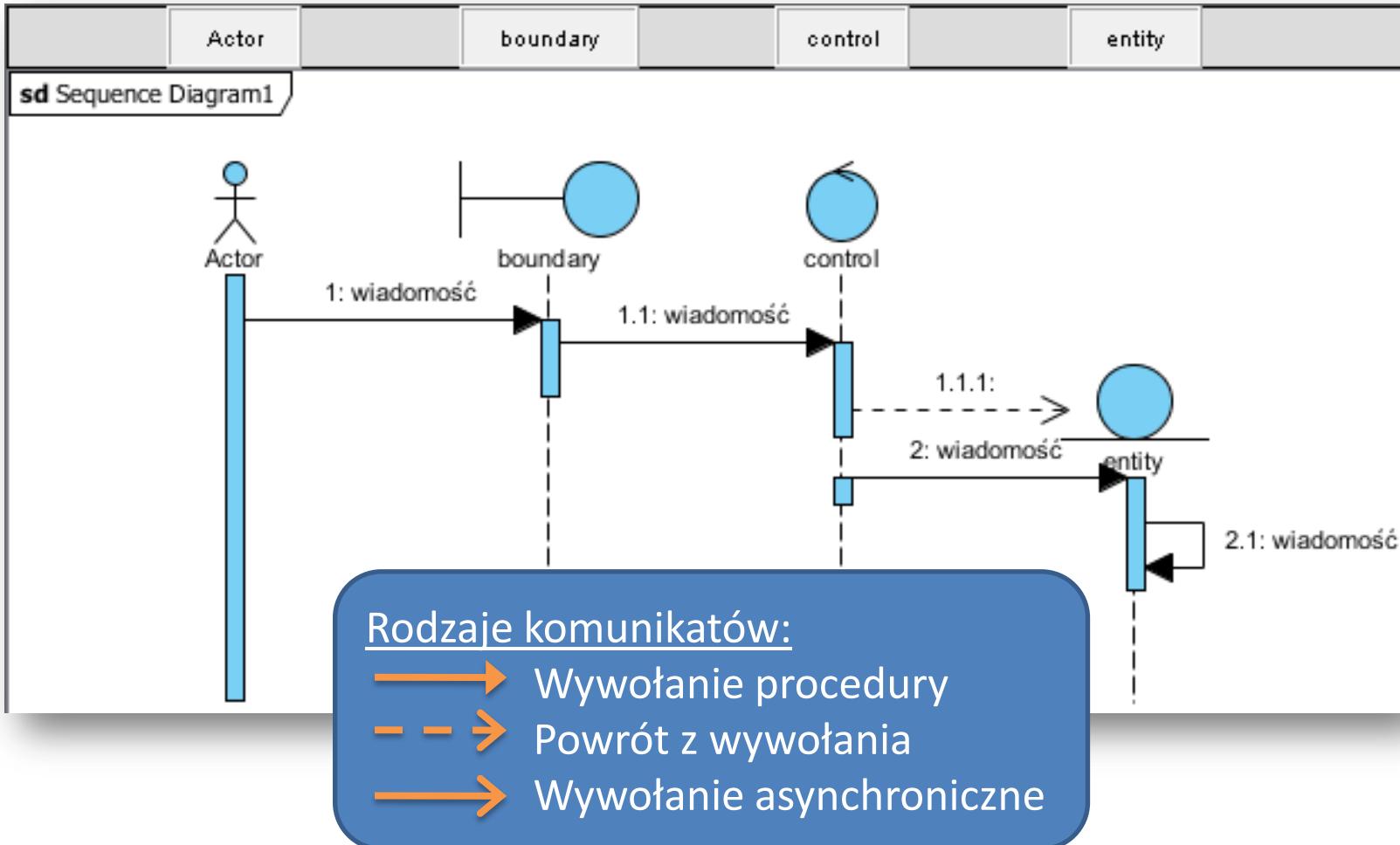
Jak rozdzielać obowiązki

- Kto ma dane potrzebne by wykonać akcję?
 - Jedna klasa posiada dane
 - Przydzielamy jej obowiązki związane z danymi
 - Wiele klas operuje na danych
 - Obowiązki dajemy jednej klasie i dodajemy relacje z pozostałymi
 - Tworzymy nową klasę i jw
 - Obowiązki nadajemy klasie typu <<control>> i jw

Diagram sekwencji

- Prezentuje interakcje poprzez wysyłanie wiadomości pomiędzy obiektami w celu osiągnięcia pożądanego celu
- Dwa wymiary:
 - Czas w dół
 - Różne obiekty w poziomie
- Składniki:
 - Obiekty
 - Przepływy
 - Opisy

Diagram sekwencji



Blok

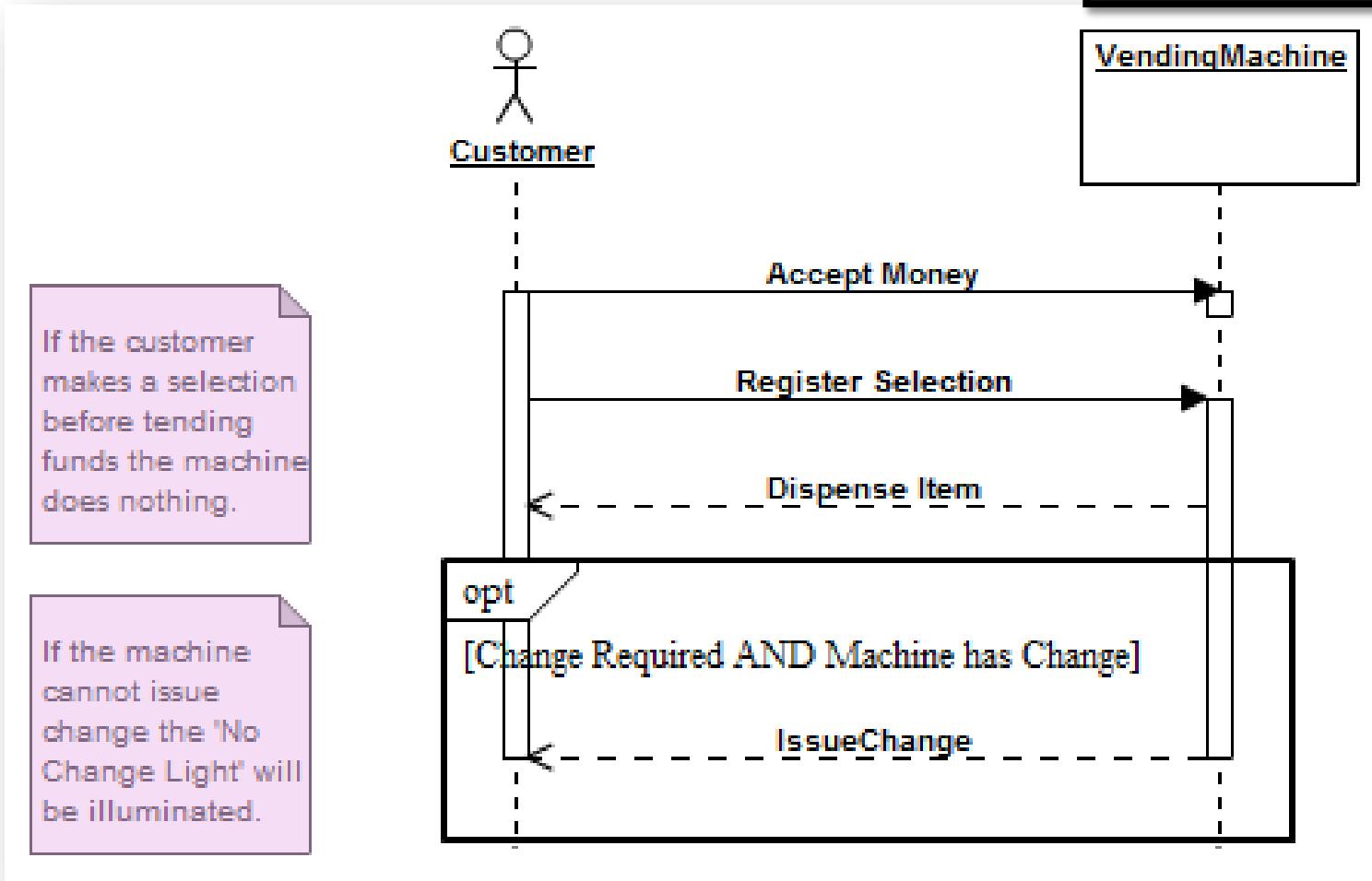
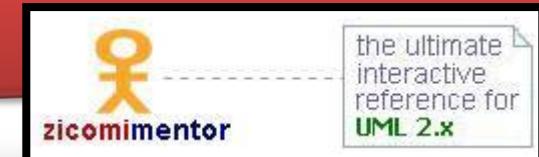
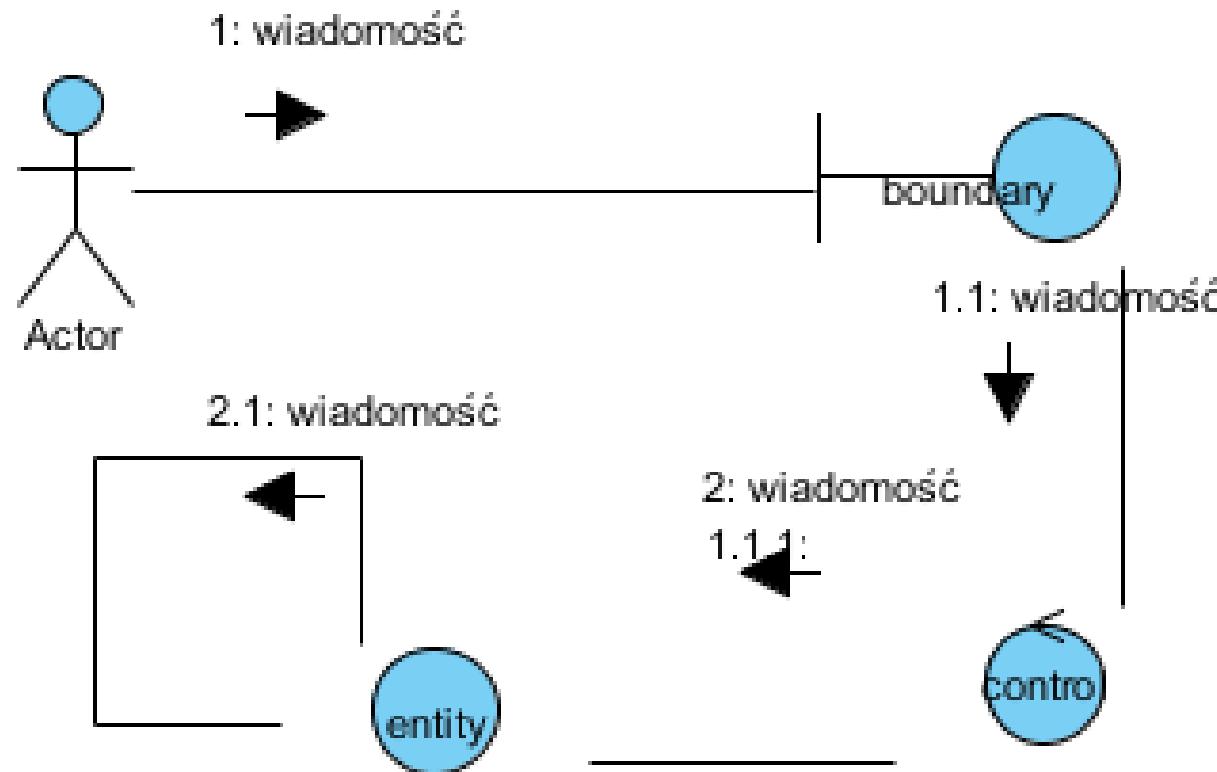


Diagram komunikacji

- Podobne w funkcji do diagramów sekwencji
- Interakcje zgromadzone są wokół ról, ich powiązania między sobą
- Pokazują związki pomiędzy obiektami pełniącymi różne funkcje
- Czas może być pokazany jedynie poprzez numerowanie

Diagram komunikacji

sd Sequence Diagram1 - Communications



Sekwencja vs komunikacja

Diagram komunikacji	Diagram sekwencji
Pokazuje relacje wraz z interakcjami.	Pokazuje dokładną sekwencję wymiany komunikatów.
Lepszy do wizualizacji wszystkich elementów wpływających na dany obiekt.	Lepszy do wizualizacji przebiegu akcji.

Diagram przeglądu interakcji

- Diagram czynności, w którym węzłami są diagramy
- Przedstawia przepływ sterowania

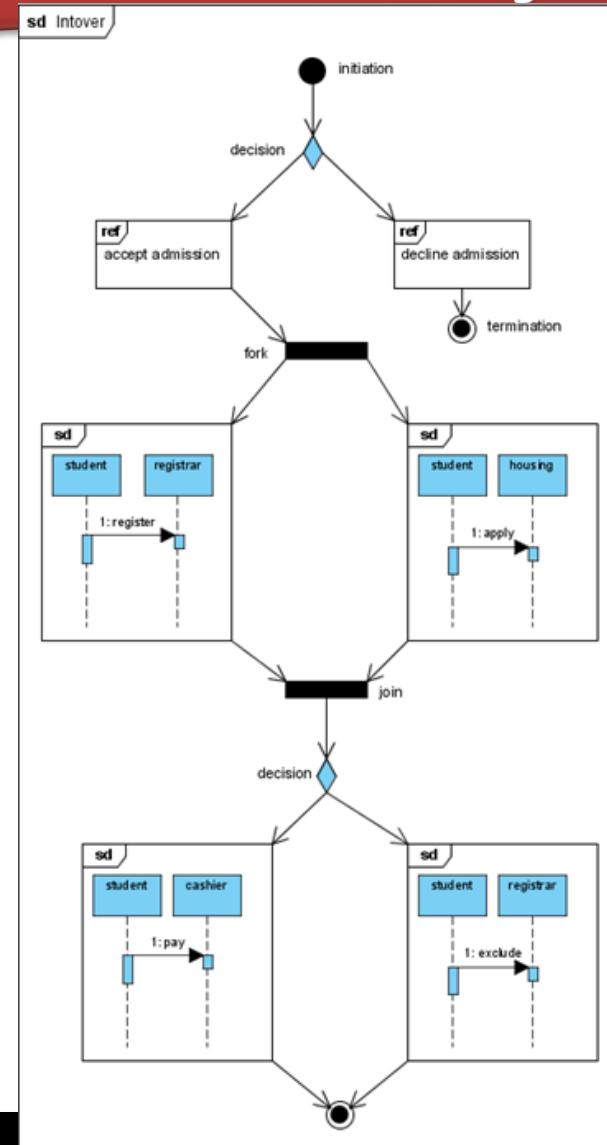
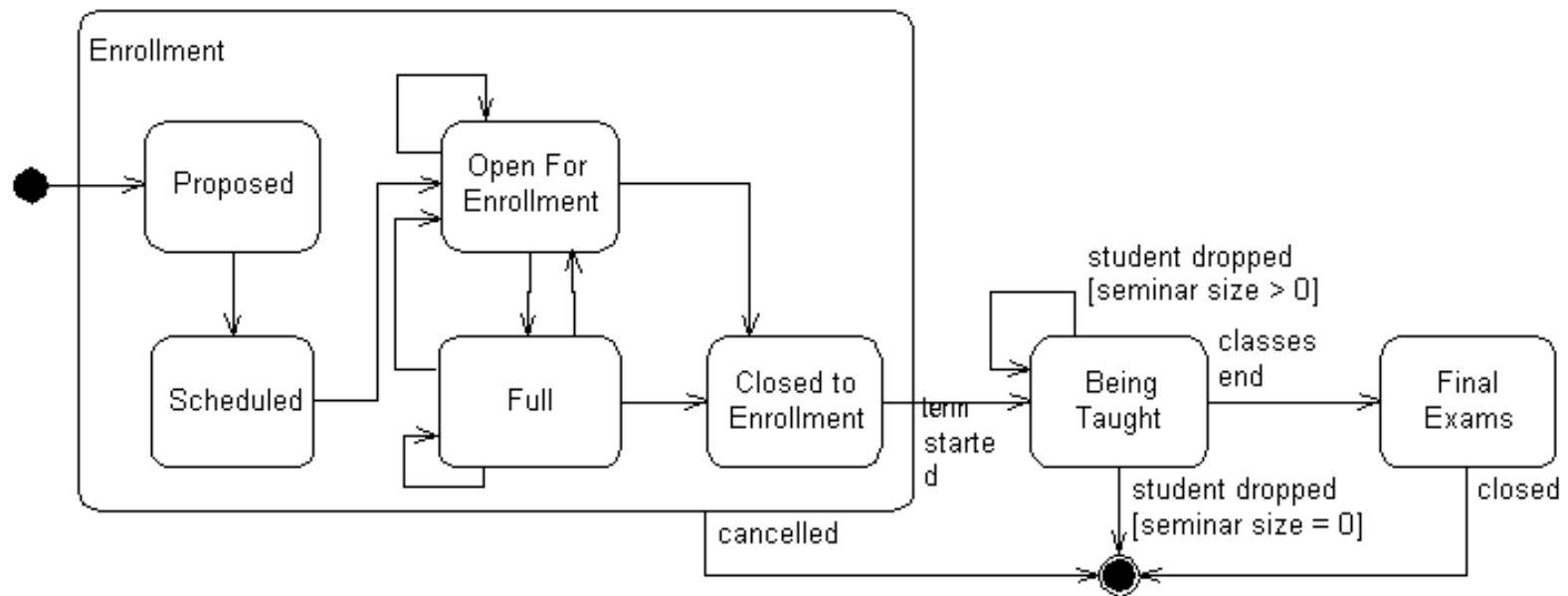


Diagram maszyny stanów

- Reprezentują zachowanie obiektów wzbudzanych poprzez otrzymywane zdarzenia.
- Opisują możliwe sekwencje stanów i akcji, przez które przechodzi element w czasie swego istnienia pod wpływem zdarzeń.
- Obiekta mi są zwykle klasy, lecz mogą też być aktorzy, metody, operacje itp...

Diagram maszyny stanowej

SEMINARIUM



KONIEC

