Modelowanie i analiza obiektowa

Wykład 2

Wprowadzenie do obiektowości

Złote myśli na dzisiejsze spotkanie

Nie pragnij, aby to co robisz było łatwiejsze.
Pragnij, abyś to Ty był lepszy.

Nie płacą ci za godzinę. Płaca ci za wartość którą wnosisz w tę godzinę.

Dyscyplina stanowi pomost pomiędzy celem a jego osiągnięciem.

Plan wykładu

- Geneza i obszary oddziaływania
- Geneza i podstawowe zasady obiektowości
- Obiektowość a notacja UML
- Diagramy klas
- Mechanizmy rozszerzalności w UML

Kryzys oprogramowania

Symptomy kryzysu oprogramowania:

- zarówno wytwarzanie, jak i utrzymywanie oprogramowania kosztuje zbyt dużo,
- oprogramowanie jest zawodne,
- współdziałanie pomiędzy produktami programistycznymi stanowi poważny problem.

Powody:

- Złożoność systemów informatycznych, syndrom współczesnych produktów, przekleństwo ciążące na większości projektów i produktów informatyki.

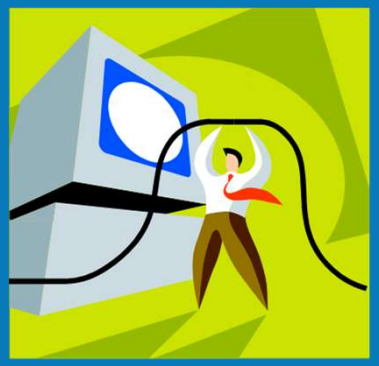
Kryzys oprogramowania c.d.

Po przekroczeniu pewnego progu złożoności człowiek przestaje “panować nad przygotowywanym produktem, ponieważ całościowe zrozumienie funkcjonowania systemu w różnych jego aspektach i na dowolnym poziomie szczegółowości z zasady przekracza jego możliwości.

Jak walczyć ze złożonością oprogramowania

Złożoność powoduje, że głównym problemem w procesie konstrukcji produktów informatycznych stał się człowiek (analityk, projektant, programista, ...) z jego różnymi uwarunkowaniami fizycznymi, psychologicznymi i mentalnymi.

Wniosek: Technologie komputerowe powinny być bardziej zorientowane na ludzi, niż na maszyny.



Geneza obiektowości

Obiektowość jest ideą, która bierze początek z zaobserwowanych wad istniejącego opisu świata i podaje receptę, jak te wady usunąć, a więc przede wszystkim stara się o uzyskanie jak najmniejszej luki pomiędzy myśleniem o rzeczywistości (dziedzinie problemowej) a myśleniem o danych i procesach, które zachodzą na danych.

Mentalna percepcja świata rzeczywistego

Model pojęciowy

Schemat struktury danych

Nowa nadzieja?

- **Obiektowość** jest nadzieją na opanowanie kryzysu oprogramowania i zredukowanie jego złożoności.

  Środki do walki ze złożonością oparte są na zasadach ***dekompozycji*** i ***abstrakcji***:

  - **Dekompozycja** - rozdzielenie złożonego problemu na niezależnie rozpatrywane podproblemy.
  - **Abstrakcja –** eliminacja lub ukrycie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji. Wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzanie pojęć lub symboli oznaczających takie cechy.

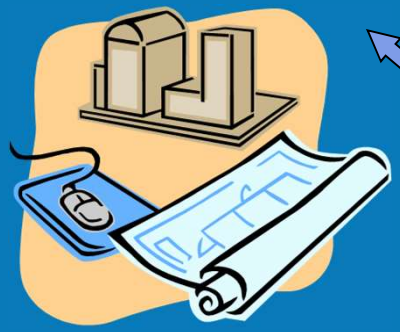
Obszary wykorzystywania idei obiektowości

- Języki programowania (Smalltalk, C++, Java, Eiffel,...)
- klasy, dziedziczenie, hermetyzacja, metody, późne wiązanie.
- Bazy danych i składy trwałych obiektów (standard ODMG-93, ObjectStore, O2, Poet, Versant, ...)
- przeniesienie obiektowych technologii programowania na grunt baz danych.
- Współdziałanie systemów heterogenicznych (OMG CORBA, OLE/DCOM/ActiveX)
- Obiekty i klasy jako podstawa wymiany informacji pomiędzy systemami.
- Inne: biblioteki oprogramowania, grafika, miary i oceny oprogramowania, re-inżynieria biznesu (BPR)

Podstawowe zasady obiektowości

Abstrakcja

Modularyzacja



Hermetyzacja

Hierarchizacja

Abstrakcja

- Eliminacja lub ukrycie mniej istotnych szczegółów rozważanego przedmiotu lub mniej istotnej informacji.
- Wyodrębnianie cech wspólnych i niezmiennych dla pewnego zbioru bytów i wprowadzanie pojęć lub symboli oznaczających takie cechy.
- Abstrakcja definiuje granice zależne od perspektywy obserwatora.
- Abstrakcja jest zależna od perspektywy oraz dziedziny problemowej, to co jest istotne w jednym kontekście, może być nieważne lub mało ważne w innym.

Przykłady abstrakcji

- Student
- Profesor
- Kurs
- Ruchomy pojazd silnikowy, transportujący ludzi z miejsca na miejsce.
- Urządzenie do bezprzewodowego odbioru sygnałów

Hermetyzacja – ukrywanie informacji

- Zasada inżynierii oprogramowania (Parnas, 1972): programista ma wiedzieć o obiekcie programistycznym tyle, ile potrzeba, aby go efektywnie użyć. Wszystko, co może być przed nim ukryte, powinno być ukryte.
- Hermetyzacja i ukrywanie informacji jest podstawą pojęć **Modułu**, **Klasy** i **ADT** (Abstrakcyjny Typ Danych).

Hermetyzacja

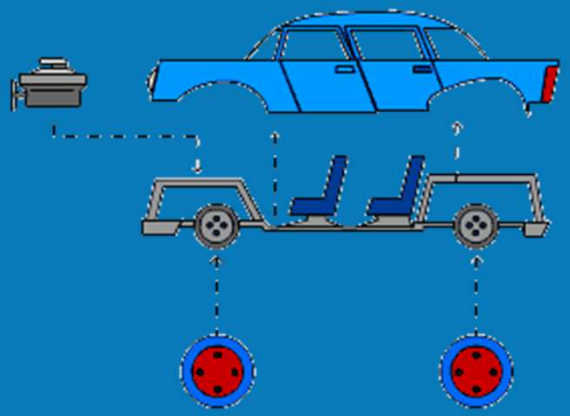
Zgromadzenie elementów struktury i implementacji obiektu w postaci jednej manipulowalnej bryły;

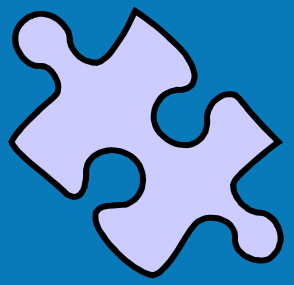
oddzielenie specyfikacji obiektu od jego implementacji; Rozróżnienie pomiędzy interfejsem do obiektu opisującym co obiekt robi, a implementacją definiującą, jak jest zbudowany i jak robi, to co ma zrobić

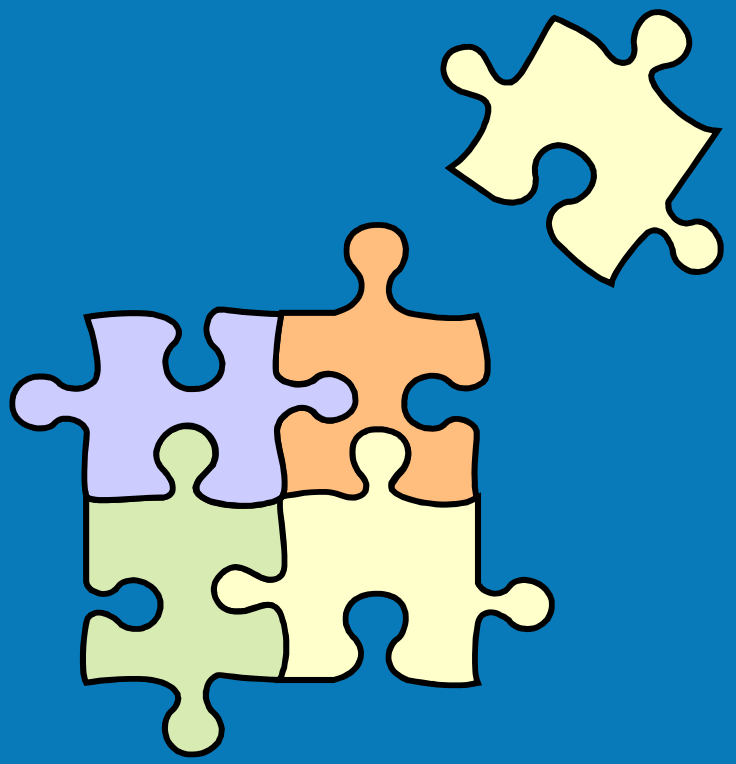
Hermetyzacja pośrednio oznacza także **ukrycie** struktury i implementacji obiektu. Tę własność określa się jako ukrywanie informacji. Hermetyzacja i ukrywanie informacji są różnymi pojęciami, choć mocno powiązanymi.

Modularyzacja - dekompozycja

Rozdzielenie złożonego zagadnienia na małe łatwiejsze do zarządzania fragmenty.



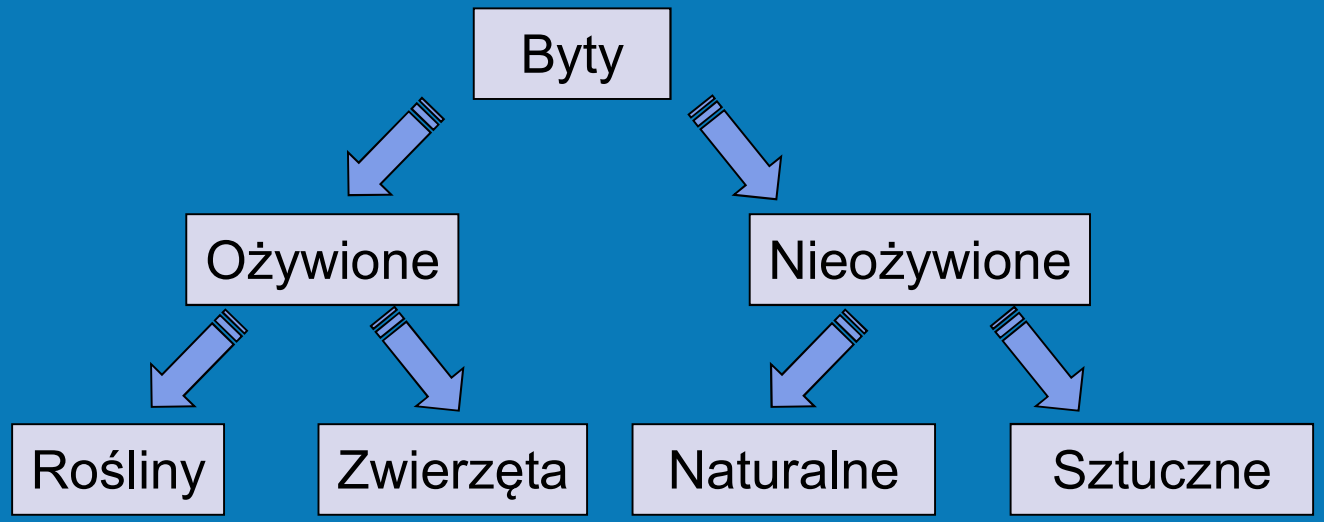




Hierarchizacja

Porządkowanie (szeregowanie) abstrakcji w strukturę drzewiastą.

Rodzaje: hierarcha agregacji, klas, dziedziczenia, typów
(Słownik terminów obiektowości, Friesmith, 1995)



Hierarchizacja

- Hierarchizacja organizuje elementy według określonego porządku, np. wg. złożoności, odpowiedzialności.
- Hierarchizacja daje w wyniku podział taksonomiczny.
- Wykorzystanie hierarchizacji daje możliwość wykrywania podobieństw i różnic pomiędzy elementami.
- Elementy na tym samym poziomie hierarchii powinny znajdować się również na tym samym poziomie abstrakcji.

Podstawowe pojęcia obiektowości

- Obiekt
  - struktura danych, występująca łącznie z operacjami na niej dozwolonymi do wykonywania, odpowiadająca bytowi wyróżnialnemu w analizowanej rzeczywistości.
- Tożsamość obiektu
  - wewnętrzny identyfikator, który pozwala na odróżnienie go od innych obiektów
- Stan i zachowanie
- Klasa
  - Zgrupowanie własności obiektów o tych samych charakterystykach (abstrakcja obiektu)
- Dziedziczenie
  - Wielokrotne użycie tego, co wcześniej zostało zrobione: definiowanie klas, które mają wszystkie cechy zdefiniowane wcześniej (z nadklasy/nadklas) plus cechy nowe.
- Polimorfizm
  - Wybór nazwy dla operacji jest określony wyłącznie semantyką operacji. Decyzja o tym, która metoda implementuje daną operację, zależy od przynależności obiektu do odpowiedniej klasy

Obiekt

- **Nieformalnie**, obiekt jest to byt obserwowalny w rzeczywistości (jej wycinku), koncepcyjny obraz tej rzeczywistości lub jednostka oprogramowania.
- **Formalnie**, obiekt jest jednostką z dobrze zdefiniowanymi granicami, który hermetyzuje stan (ang. state) oraz zachowanie (ang. behavior).

Obiekt

- Obiekt posiada informacje i wie jak wykonać określone zdania
- Obiekt może być złożony, tj. może składać się z innych obiektów.
- Pojęcie obiektu sprzyja lepszemu rozumieniu modelowanego świata rzeczywistego - byty ze świata rzeczywistego odpowiadają obiektom w programie.

Obiekt

- Obiekt to zamknięty fragment oprogramowania (dana, procedura, moduł, dokument, okienko dialogu,...), którym można operować jak zwartą bryłą: usuwać, wyszukiwać, wiązać, kopiować, blokować, indeksować, ...
- Obiekt ma przypisany typ, tj. wyrażenie językowe, które określa jego budowę (poprzez specyfikację atrybutów) oraz ogranicza kontekst, w którym odwołanie do obiektu może być użyte w programie.
- Obiekt może być powiązany z innymi obiektami związkami skojarzeniowymi, odpowiadającymi relacjom zachodzącym między odpowiednimi bytami w dziedzinie problemowej.

Tożsamość obiektu

identity

Może się zdarzyć, że z punktu widzenia naszych obserwacji (tj. stanu obiektu) dwa obiekty są nieodróżnialne. Niemniej jednak są to dwa różne obiekty.

Obiekt jest wyróżnialny w otaczającym nas świecie poprzez swoje istnienie, a nie poprzez jakąkolwiek wartość, która odróżnia go od innych obiektów.

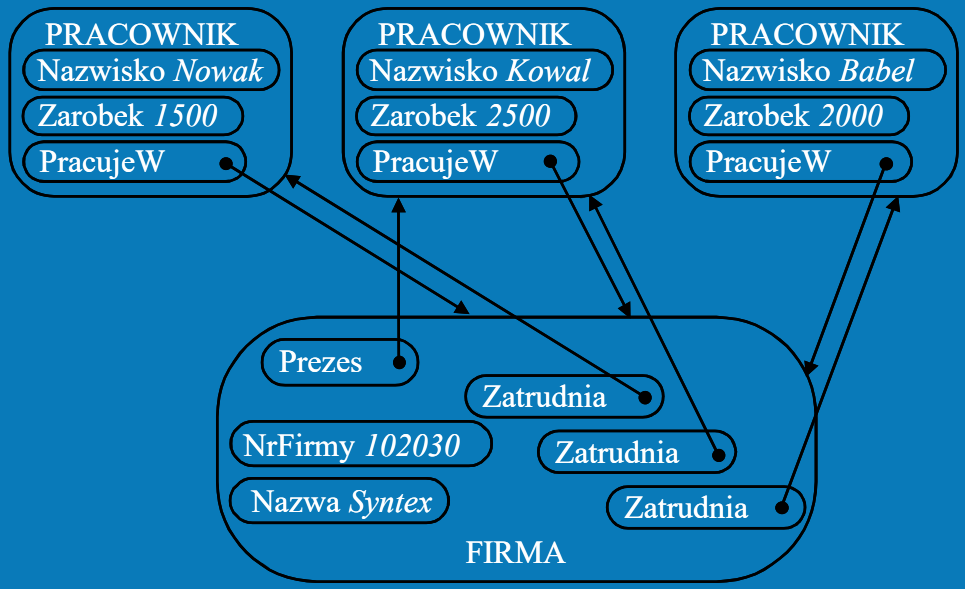
Innymi słowy: to, że jakaś struktura danych posiada takie same wartości (atrybutów) nie oznacza to, że mamy do czynienia z tym samym elementem.
Istnieje przecież możliwość spotkania dwóch Janów Kowalski urodzonych tego samego dnia w Polsce ale nie oznacza to, że jest to jedna i ta sam osoba.

Własności obiektu

- Tożsamość, która odróżnia go od innych obiektów. Tożsamość obiektu jest niezależna zarówno od wartości atrybutów obiektu, jak i od lokacji bytu odwzorowywanego przez obiekt w świecie rzeczywistym czy też lokacji samego obiektu w przestrzeni adresowej komputera. (W praktyce: tożsamość = trwały wewnętrzny identyfikator obiektu)
- Stan, który może zmieniać się w czasie (bez zmiany tożsamości obiektu). Stan obiektu w danym momencie jest określony przez aktualne wartości jego atrybutów i powiązań z innymi obiektami.
- Obiekt ma przypisane zachowanie, tj. zestaw operacji, które wolno stosować do danego obiektu.
- Obiekt może być złożony, tj. może składać się z mniejszych obiektów.
- Obiekt może być powiązany z innymi obiektami związkami skojarzeniowymi.
- Obiekt ma przypisany typ, tj. wyrażenie językowe, które określa dopuszczalną budowę obiektu oraz ustala operacje, które wolno wykonywać na obiekcie.

Powiązania pomiędzy obiektami

links, pointers, relationships



W obiektowych językach programowania, metodykach i bazach danych możliwe jest tworzenie bezpośrednich powiązań prowadzących od jednego obiektu do innego. Powiązanie jest daną zawierającą identyfikator obiektu. Unika się tu pojęcia wskaźnika, na rzecz czegoś “bardziej abstrakcyjnego” - powiązania.

Powiązania pomiędzy obiektami

links, pointers, relationships

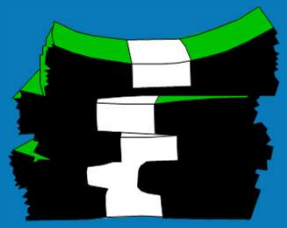
Zalety powiązań:

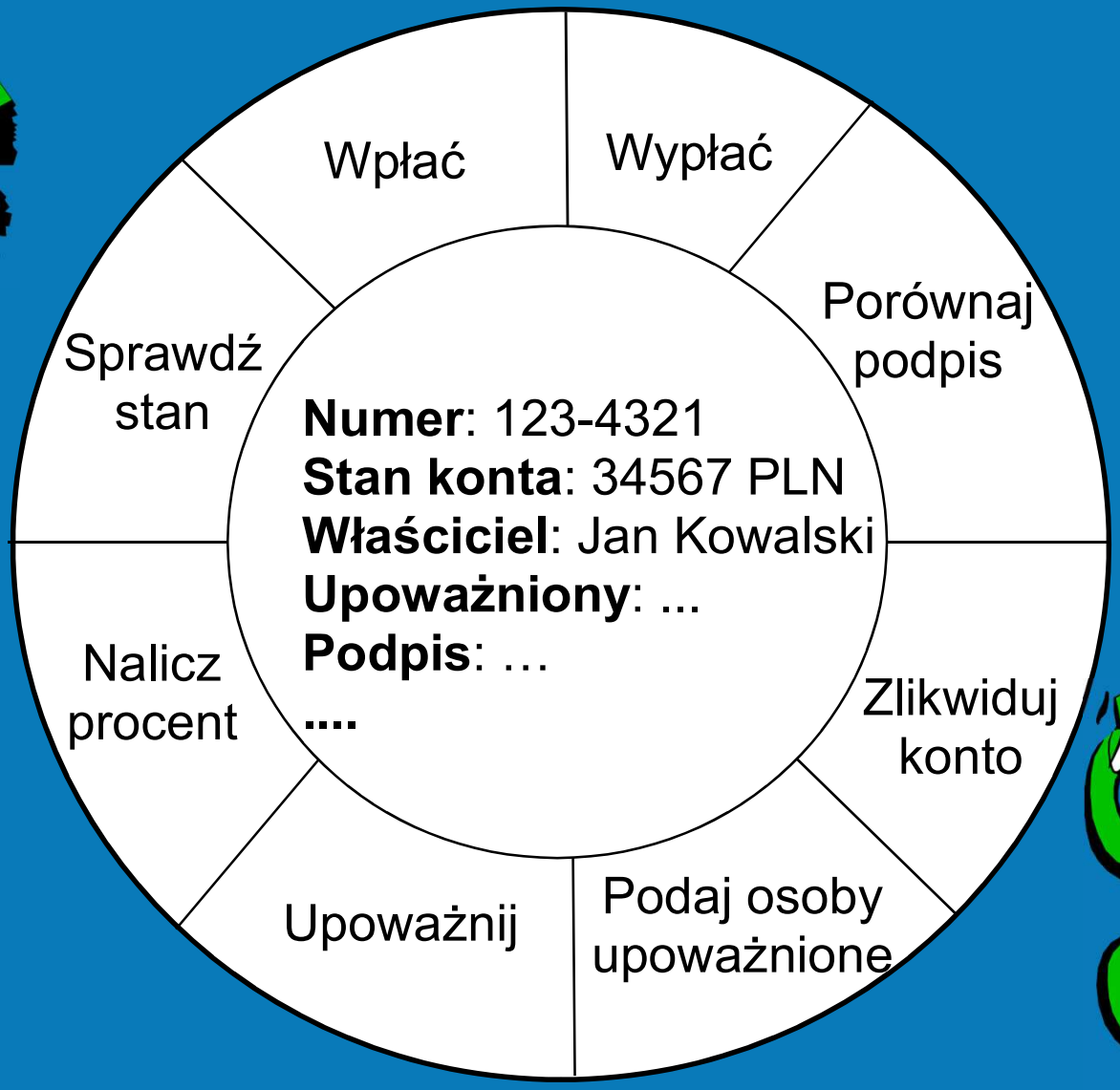
- naturalne odwzorowanie semantycznych związków między obiektami,
- łatwe nawigowanie dzięki wyrażeniom ścieżkowym,
- zwiększenie szybkości działania.

Wady:

- zwiększona “sztywność” struktury danych,
- możliwość utraty spójności wskutek tzw. “zwisających” (*dangling*) wskaźników,
- możliwość naruszenia reguł hermetyzacji.

Przykład obiektu

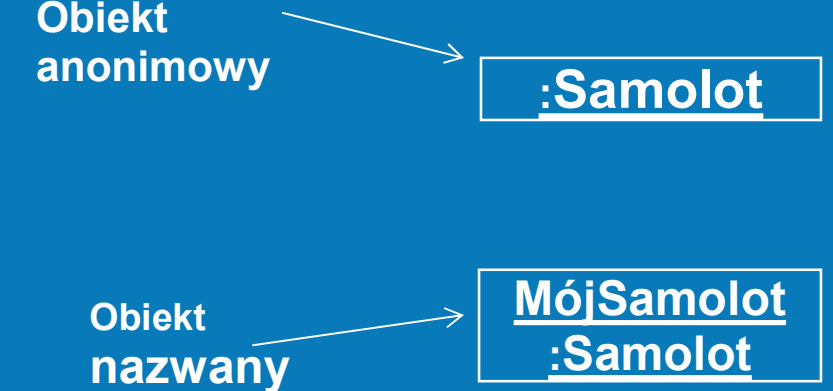




Obiekt
KONTO

Reprezentacja obiektów w UML

Obiekt jest reprezentowany przez prostokąt w środku którego znajduje się podkreślona nazwa



Klasa

Klasa jest:

- opisem grupy obiektów,
- miejscem przechowywania własności (inwariantów) grupy obiektów,
- abstrakcyjną definicją obiektu

Obiekt jest:

wystąpieniem (instancją) klasy.

Klasa

- Klasa jest miejscem przechowywania tych własności grupy obiektów, które są niezmienne (inwariantne) dla wszystkich członków grupy.

- Klasa nie jest zbiorem obiektów. Stosunek klasa/podklasa oznacza, że obiekty podklasy posiadają wszystkie inwarianty nadklasy, plus ewentualnie swoje inwarianty. Np. klasa Student ma wszystkie inwarianty klasy Osoba, plus inwarianty własne.

Klasa, cd..

Najważniejsze inwarianty to:

- Nazwa, czyli językowy identyfikator klasy obiektu,
- Typ, czyli struktura (budowa) obiektu – poprzez atrybuty,
- Metody, czyli operacje, które można wykonać na obiekcie.

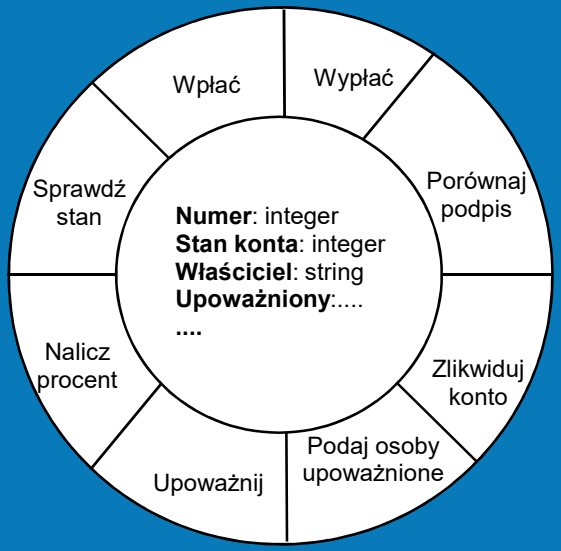
Możliwe są inne inwarianty:

- Zdarzenia lub wyjątki, które mogą zachodzić w operacjach na obiekcie,
- Obsługa zdarzeń lub wyjątków (reguły aktywne),
- Lista eksportowa określająca, co jest dostępne z zewnątrz,
- Ograniczenia, którym może podlegać obiekt klasy

Klasa i obiekt

Zwykle, mamy do czynienia z wieloma obiektami tej samej klasy, np. z wieloma kontami. Niecelowe jest, aby każdy z takich obiektów przechowywał w sobie własną kopię metod lub informacji o swoim typie (budowie). Ta informacja jest przechowywana w jednym miejscu, w klasie.

Klasa wszystkich kont



import inwariantów

Obiekty KONTO

Numer: 1234321
Stan konta: 34567
Właściciel: Jan Kowalski
Upoważniony:
....

Numer: 1234567
Stan konta: 454545
Właściciel: Adam Nowak
Upoważniony:
....

Klasa w UML - notacja

- Trzy pola: nazwy, atrybutów i metod.
- Możliwe są różne poziomy szczegółowości

Okno	Okno	Okno	Okno
	rozmiar czy_widoczne	rozmiar czy_widoczne	rozmiar: Obszar czy_widoczne: Boolean
		wyswietl() schowaj()	wyswietl() schowaj()
Tylko nazwa klasy	Nazwa + pole atrybutów	Nazwa + pole atrybutów + pole metod	Nazwa + pole atrybutów (z typami) + pole metod

Pole nazwy klasy:

- stereotyp nazwa_ klasy lista_wart_etyk

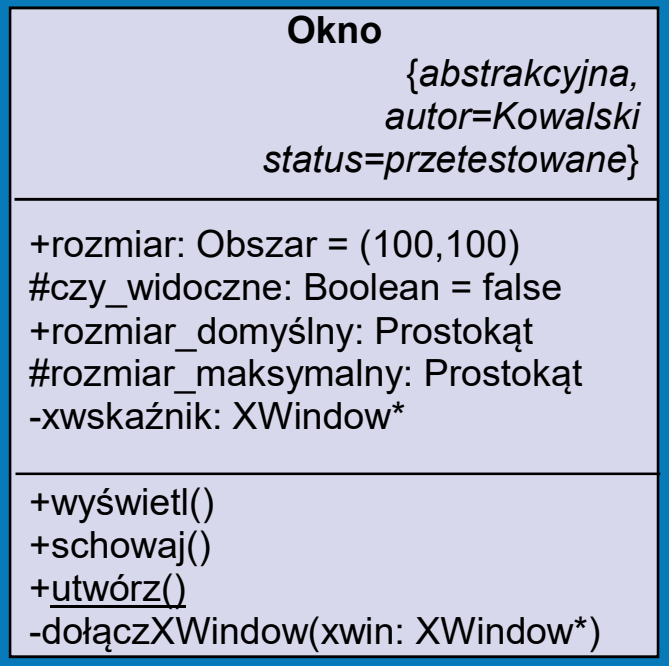
Pole atrybutów:

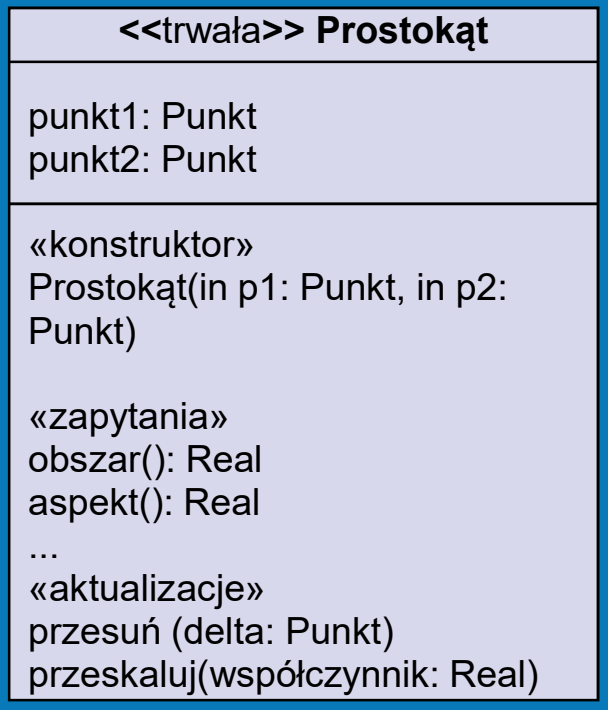
- stereotyp dostępność nazwa_atrybutu : typ = wart_początkowa lista_wart_etyk

Pole metod:

- stereotyp dostępność nazwa_metody (lista_arg) : typ_wart_zwracanej lista_wart_etykt

Klasa w UML – notacja cd.





Klasa w UML – notacja cd.

Dostępność:

- \+ publiczna
- \- prywatna
- \# chroniona

Lista_arg:

rodzaj nazwa_arg : typ = wart_początkowa

rodzaj definiuje sposób, w jaki metoda korzysta z danego argumentu:

in: metoda może czytać argument, ale nie może go zmieniać

out: może zmieniać, nie może czytać

inout: może czytać i zmieniać

Wszystkie elementy specyfikacji klasy za wyjątkiem nazw: klasy, atrybutu, metody są opcjonalne.

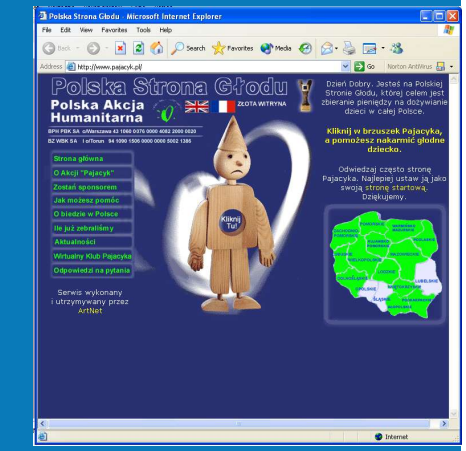
Nazwa klasy to z reguły rzeczownik w liczbie pojedynczej.

Stereotypy (<<stereotyp>>) będą omówione w dalszej części wykładu.

Atrybut - definicja

Atrybut jest to nazwana własność klasy, która opisuje zakres wartości jakie instancja tej klasy (czyli obiekt) może przechowywać

Okno
rozmiar czy_widoczne rozmiar_max



Atrybut posiada typ, który określa jego rodzaj i jest ograniczeniem na kontekst, w którym dany atrybut może się pojawić. Typy mogą być proste i złożone

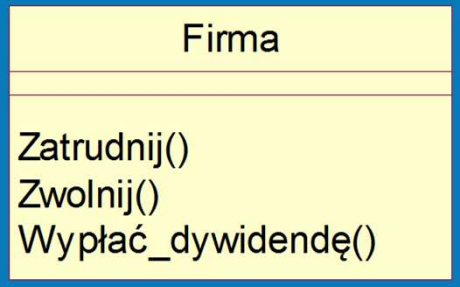
Okno
rozmiar rozwierane uchylne

Operacja

- Operacja jest implementacją usługi, jaką udostępniają instancje klasy.
- Klasa może posiadać określoną liczbę operacji lub nie posiadać żadnej

zatrudnij
zwolnij
wypłać dewidendę

} możliwe operacje na obiektach klasy Firma

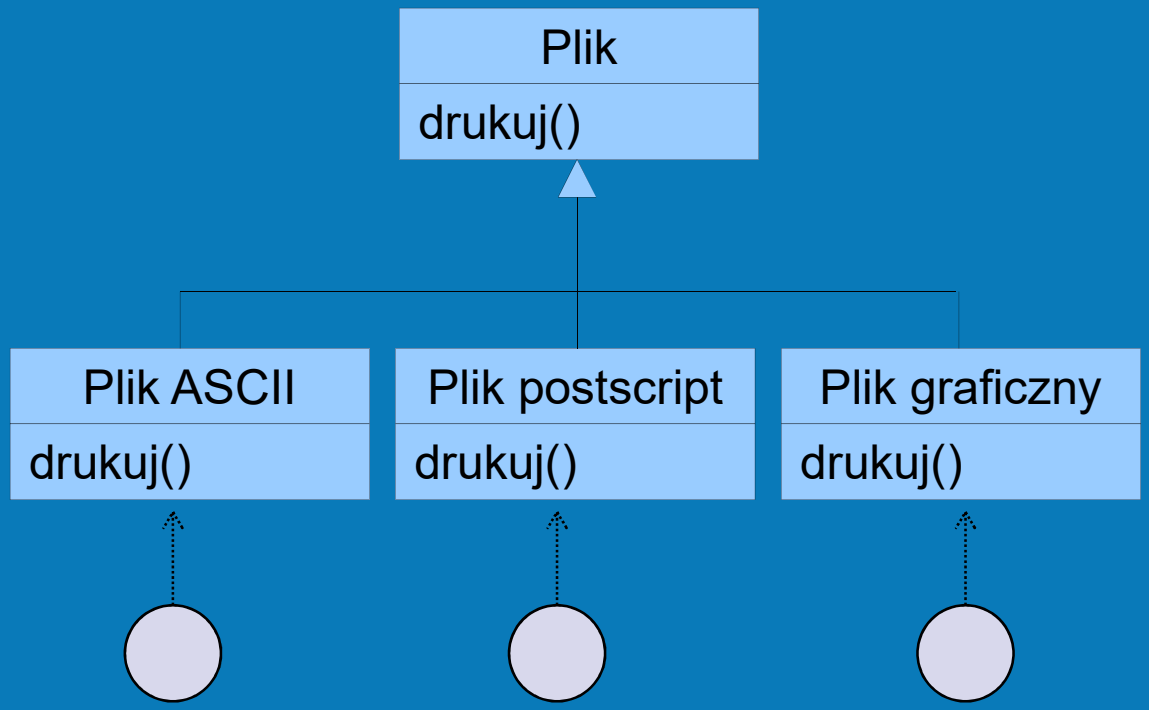


Operacja vs. metoda

- Operacja jest funkcją, która może być zastosowana do obiektu.
- Operacja jest własnością klasy obiektów, ponieważ jest przechowywana w klasie.
- Wszystkie obiekty, będące członkami danej klasy, podlegają tym samym operacjom.
- Dana operacja może być stosowana do obiektów wielu różnych klas, połączonych związkiem generalizacji-specjalizacji.
- Metoda jest implementacją operacji w jednej z klas połączonych związkiem generalizacji-specjalizacji, co oznacza, że może być wiele metod implementujących daną operację.

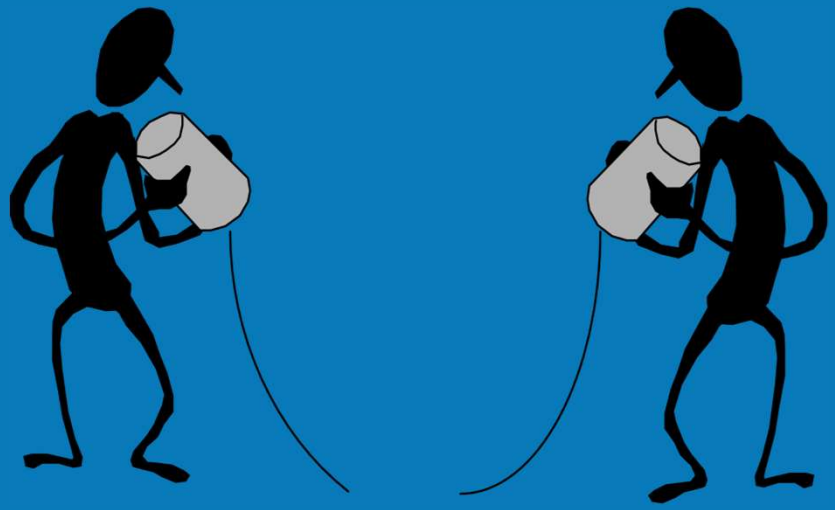
Operacja a metoda - polimorfizm

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



Komunikat

Komunikat jest wyrażeniem językowym (z nazwą operacji) skierowanym do obiektu i wywołującym jedną z operacji związanych z klasą , której członkiem jest obiekt; oznacza to wywołanie metody implementującej daną operację, najbliższej obiektowi w sensie hierarchii dziedziczenia.



Po wykonaniu operacji obiekt, który otrzymał komunikat może zwrócić odpowiedź do obiektu, który go wysłał. **Odpowiedź nie jest komunikatem**.

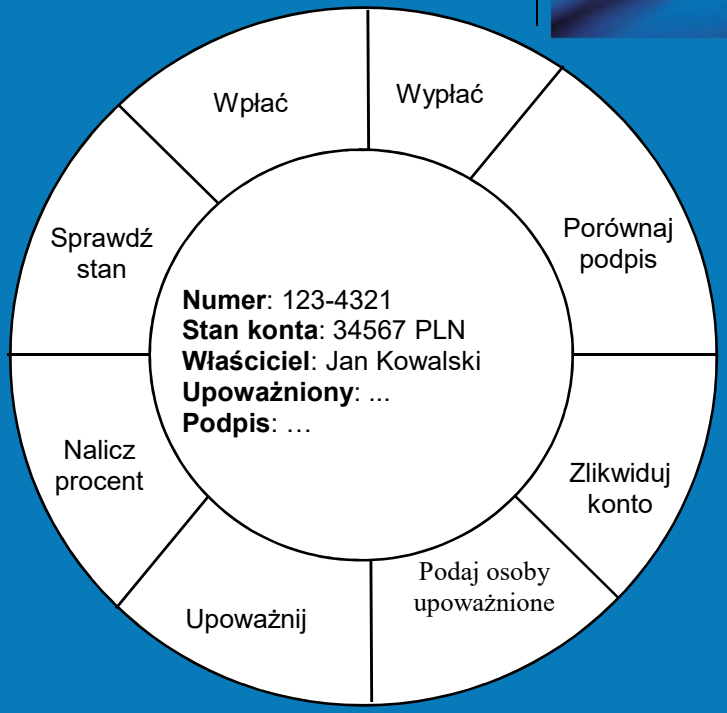
Komunikat

- Komunikat jest wysyłany z jednego obiektu do drugiego.
- Komunikat może mieć zero, jeden lub więcej parametrów.
- Komunikat może zmienić stan obiektu, który go otrzymał.
- Postać komunikatu nie zależy od implementacji obiektu, natomiast wykonanie komunikatu zależy od implementacji obiektu.

Komunikat cd.

Wypłać 1000 PLN

OK, wypłaciłem



Graj

?

Cccco proszę...?

Komunikat a wołanie funkcji

- **Komunikat**: obiekt-adresat poprzedza wywołanie operacji: ***obiekt.operacja (arg1, arg2,...)*** - obiekt jest tu domyślnym argumentem
- **Wołanie funkcji**: obiekt jest komunikowany jako parametr: ***funkcja (obiekt, arg1, arg2,...)***

Nie jest to wyłącznie różnica syntaktyczna

Komunikat a wołanie funkcji

- Dla metody, środowisko na którym działa, może zmieniać się dynamicznie (późne wiązanie metod, w odróżnieniu od wczesnego wiązania dla funkcji).
- Komunikat nie określa, *która* z metod implementujących daną operację ma być wywołana; wysłanie komunikatu do konkretnego obiektu powoduje wywołanie metody, implementującej żądaną operację, związanej z danym obiektem.
- **Przykład**: operacja *drukuj* wykonywana iteracyjnie dla zbioru plików o różnym formacie.

Komunikat a wołanie funkcji

X = dochody(emeryt)
Y = dochody(pracownik)

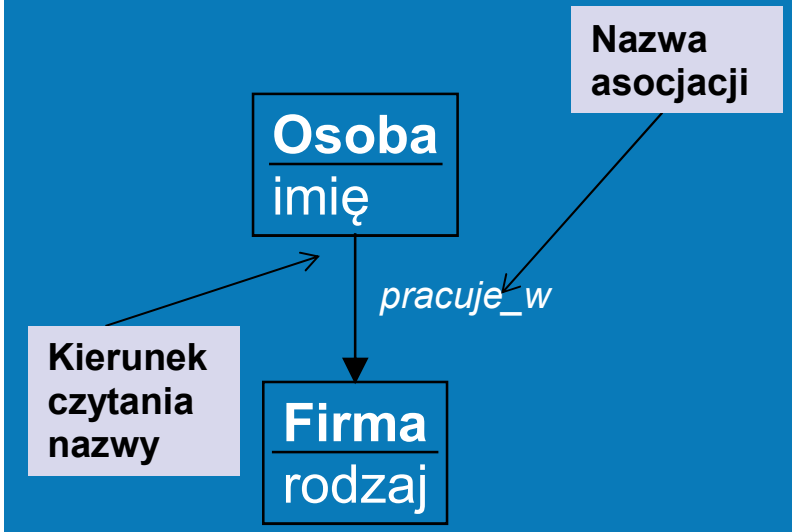
Przełączanie, związane z rodzajem obiektu, następuje w ciele funkcji *dochody*. Funkcję zwykle pisze jeden programista, na wszystkie okazje.
Każda zmiana, związana z nowym rodzajem obiektów, wymaga zmian w ciele funkcji.

X = emeryt.dochody()
Y = pracownik.dochody()

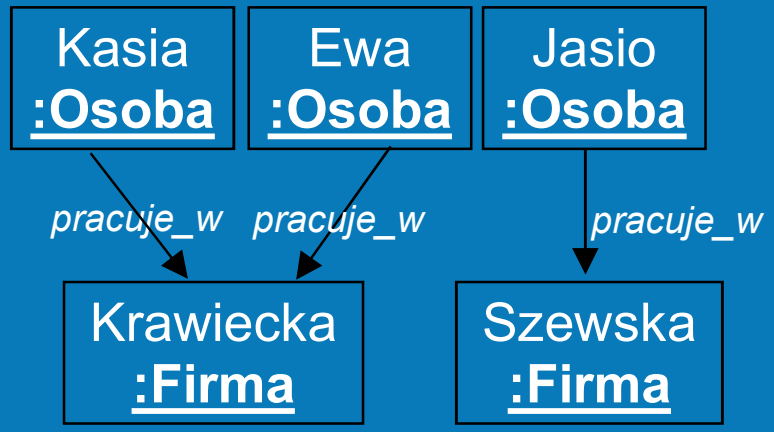
Nie ma przełączenia; za każdym razem, w zależności od rodzaju obiektu - adresata komunikatu, wywoływana jest inna metoda *dochody*.
Obie metody implementują operację *dochody.* Obie metody (i ich programiści) nie muszą nic o sobie wiedzieć.

Powiązania (ang. *relationships*) i asocjacje (ang. *associations*)

- **Powiązanie**: fizyczny lub pojęciowy związek między obiektami, odwzorowujący związek istniejący między odpowiednimi bytami w analizowanej dziedzinie przedmiotowej.
- **Asocjacja**: grupa powiązań posiadających wspólną strukturę i semantykę, czyli jest sposobem pokazania związku na diagramie klas. Powiązanie jest wystąpieniem asocjacji



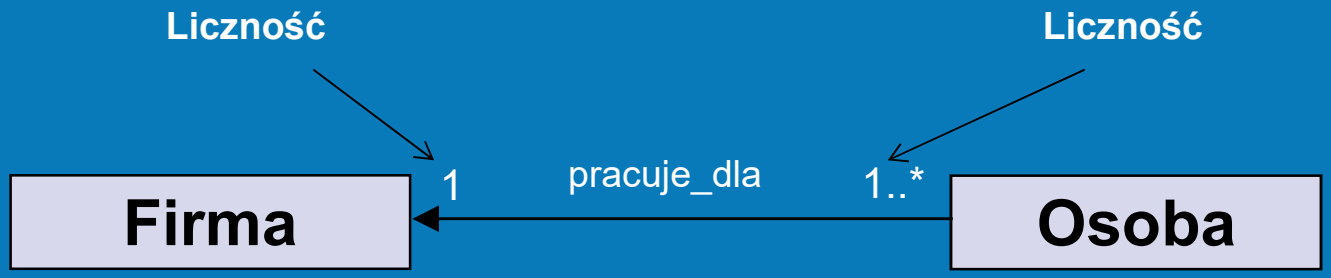
Klasy i asocjacja na diagramie klas



Obiekty i powiązania na diagramie obiektów

Liczność asocjacji

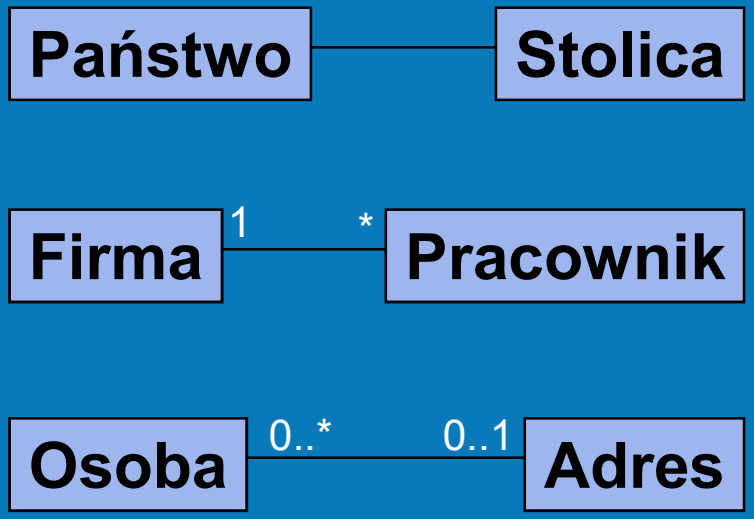
Asocjacje mogą być wyposażone w oznaczenia liczności. Liczność oznacza, ile obiektów innej klasy może być powiązane z jednym obiektem danej klasy; zwykle określa się to poprzez parę liczb (znaków), oznaczającą minimalną i maksymalną liczbę takich obiektów.



Cecha o dużym znaczeniu informacyjnym w analizie (i modelowaniu w ogóle).

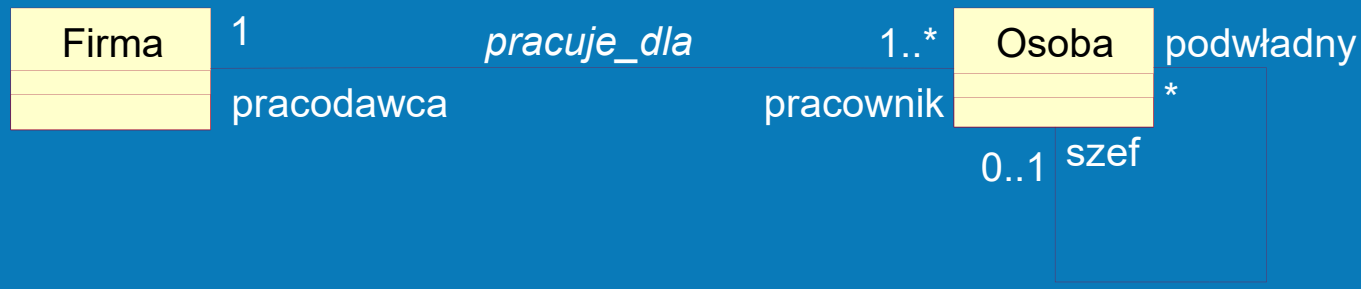
Oznaczenia liczności

UML	znaczenie
1	1
1..*	1,2,3,...
2..*	2,3,4,...
3-5	3,4,5
2,4,18	2,4,18
	1, ?
0..1	0,1
0..*	0,1,2,...
*	0,1,2,...



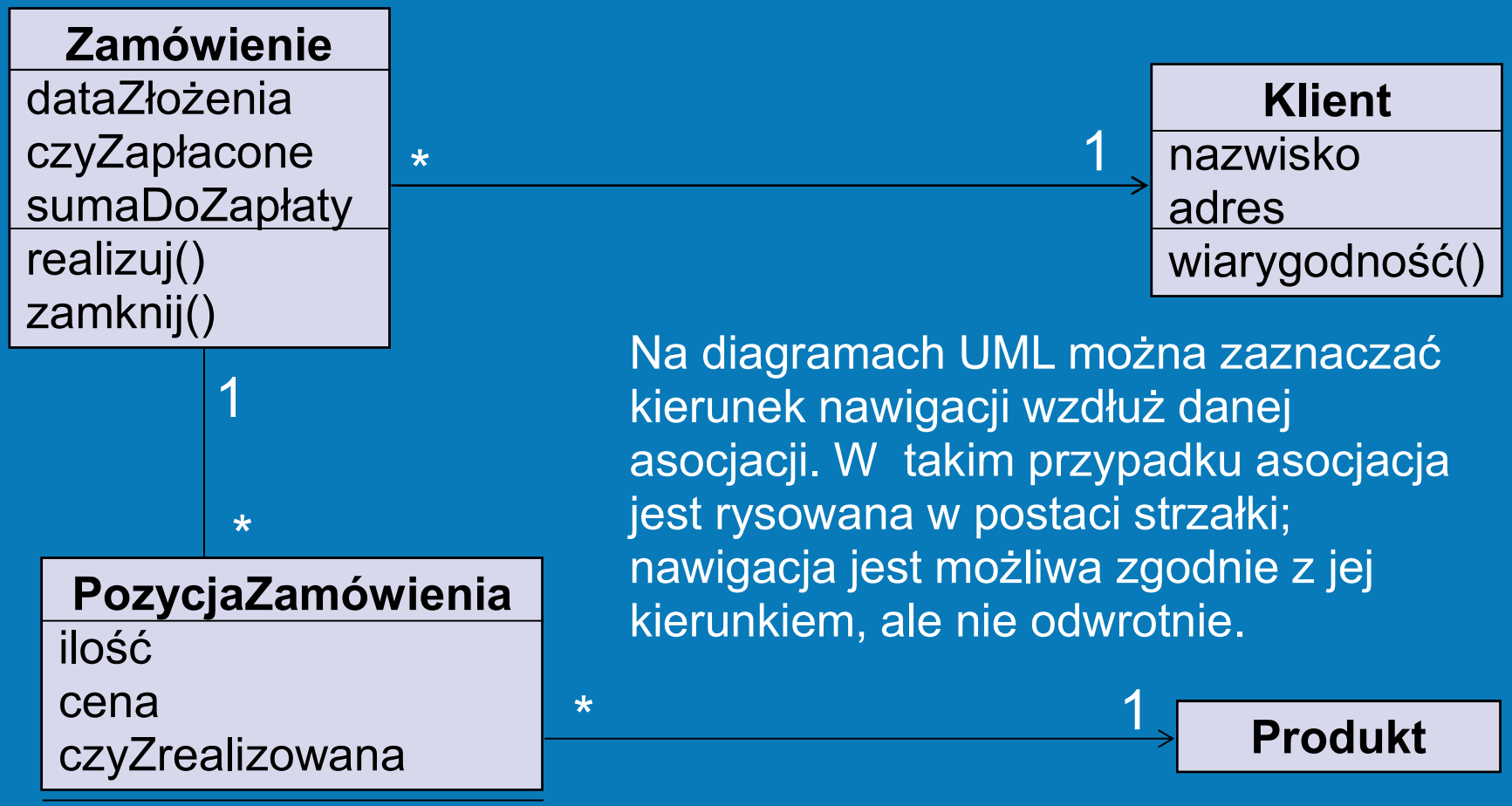
Role asocjacji

Asocjacje mogą być także wyposażone w nazwy ról (przy odpowiednich końcach), np. ***pracodawca*** **i** ***pracownik***.



Role asocjacji są niezbędne, gdy powiązania łączą obiekty tej samej klasy.

Nawigacja (ang. *navigability*) – asocjacje skierowane

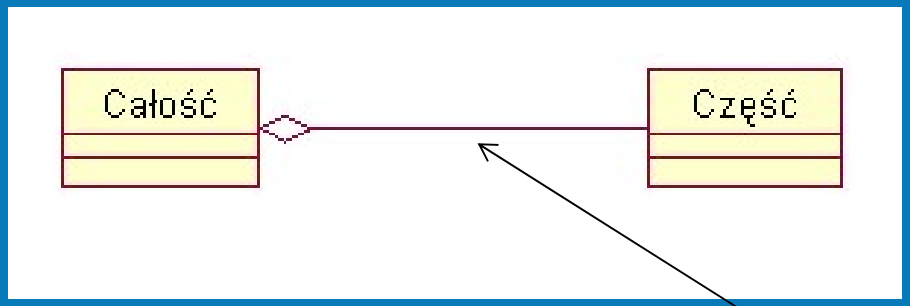


Na diagramach UML można zaznaczać kierunek nawigacji wzdłuż danej asocjacji. W takim przypadku asocjacja jest rysowana w postaci strzałki; nawigacja jest możliwa zgodnie z jej kierunkiem, ale nie odwrotnie.

W języku UML asocjacje są domyślnie nawigowalne w obu kierunkach.

Agregacja (ang. *aggregation*)

Agregacja jest szczególnym przypadkiem asocjacji wyrażającym zależność część-całość. Np. silnik jest częścią samochodu, biblioteka zawiera książki, komputer składa się z podzespołów.



Oznaczenie agregacji

Związek agregacji również może być etykietowany licznością oraz nawigacją

Agregacja (ang. *aggregation*)

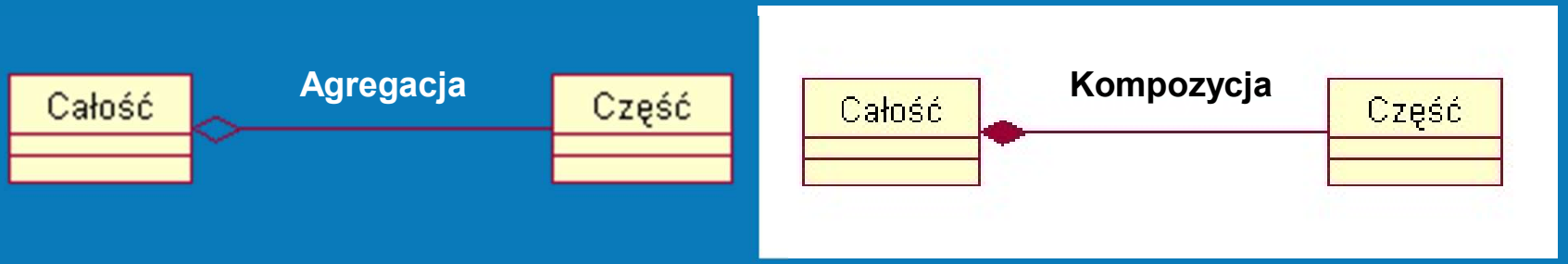
- Nie istnieje jednak powszechnie akceptowana definicja agregacji:
- P. Coad podaje przykład agregacji jako związek pomiędzy organizacją i jej pracownikami;
- dla odmiany J. Rumbaugh twierdzi, że firma nie jest agregacją jej pracowników.
- W wielu przypadkach związki agregacji są oczywiste. Jednakże wątpliwości powstają nawet w przypadku samochodu i silnika:
- na przykład – silnik może być towarem w sklepie nie związanym z żadnym samochodem,
- ponadto, czy chodzi o konkretny samochód i silnik, czy też o typ samochodu i typ silnika?

Dwa typy agregacji

- **Pojęcie agregacji jest rozumiana na dwa sposoby:**

- Jako związek część-całość pomiędzy obiektami świata rzeczywistego; np. silnik jest częścią samochodu.

- Jako pomocniczy środek do modelowania dowolnej innej sytuacji, kiedy trzeba wydzielić podobiekty w pewnych obiektach. np. informacja o ubezpieczeniach wewnątrz obiektów pracowników.

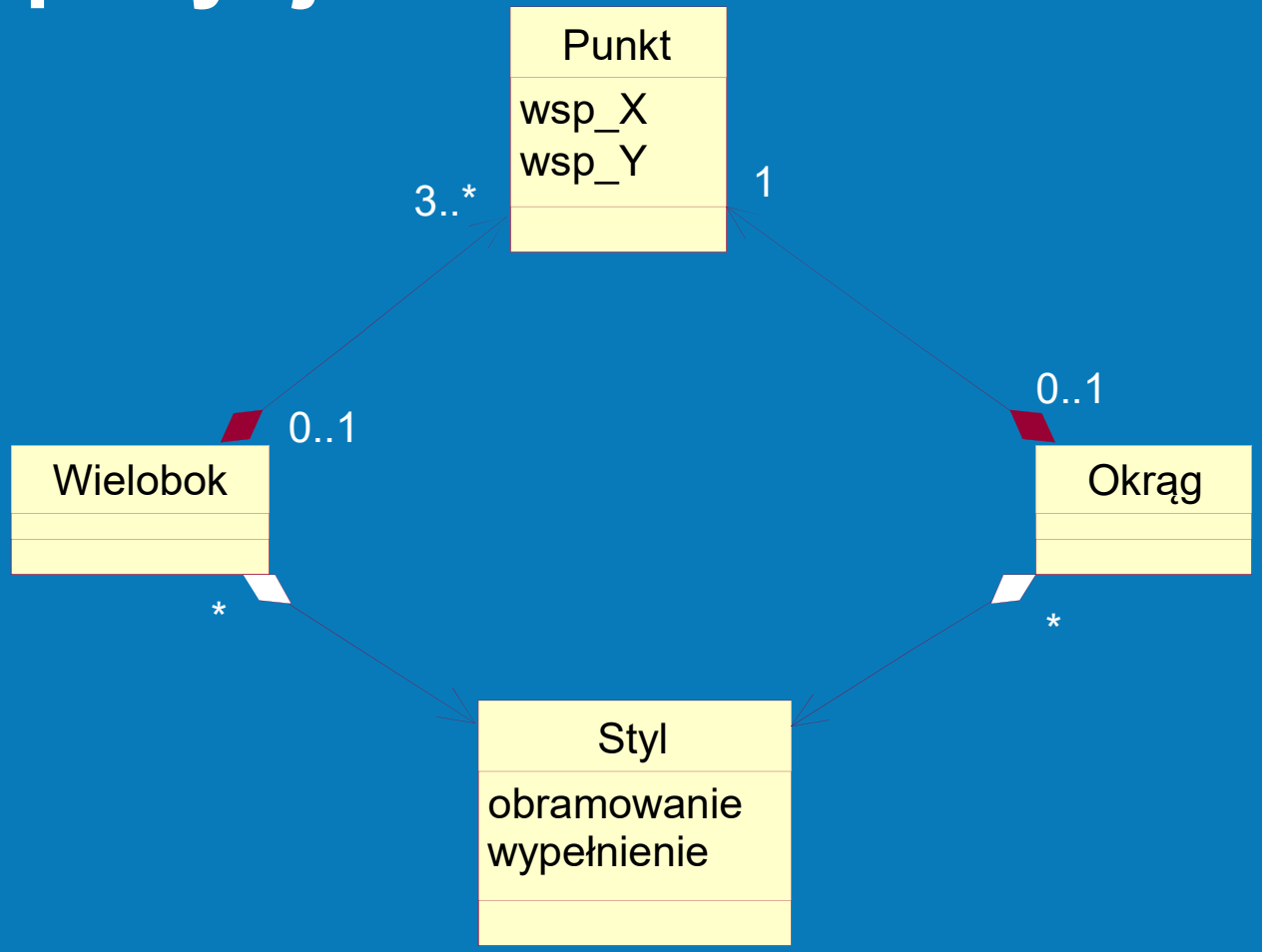


W UML te dwie sytuacje zostały rozdzielone. Pierwszą formę nazwano **kompozycją**. Kompozycja oznacza, że **cykl życiowy składowej zawiera się w cyklu życiowym całości, oraz że składowa nie może być współdzielona**.

Agregacja czy kompozycja?

- Agregacja:
  - Element stanowiący część nie jest przechowywany bezpośrednio w obiekcie całość.
  - Przechowywana jest tylko referencja do niego.
  - Skasowanie obiektu całość nie powoduje skasowania obiektu część.
  - Dodatkowo może się zdarzyć, że obiekt część jest współdzielony z innym obiektem całość.
- Kompozycja:
  - Obiekt „część” jest przechowywany bezpośrednio w obiekcie „całość”
  - Stanowi on jego integralną część, nie może być współdzielony i skasowanie całości powoduje skasowanie obiektu część.

Przykład: agregacja i kompozycja



W przedstawionym rozwiązaniu, punkt, w którym przecinają się **Okrąg** i **Wielobok,** jest odwzorowywany w dwa obiekty klasy **Punkt**.

Generalizacja czyli uogólnienie (ang. *generalization*) - dziedziczenie

Związek pomiędzy klasami, definiujący hierarchię abstrakcji, w którym jedna klasa dziedziczy z jednej bądź wielu nadklas.

- Dziedziczenie pojedyncze
- Dziedziczenie wielokrotne

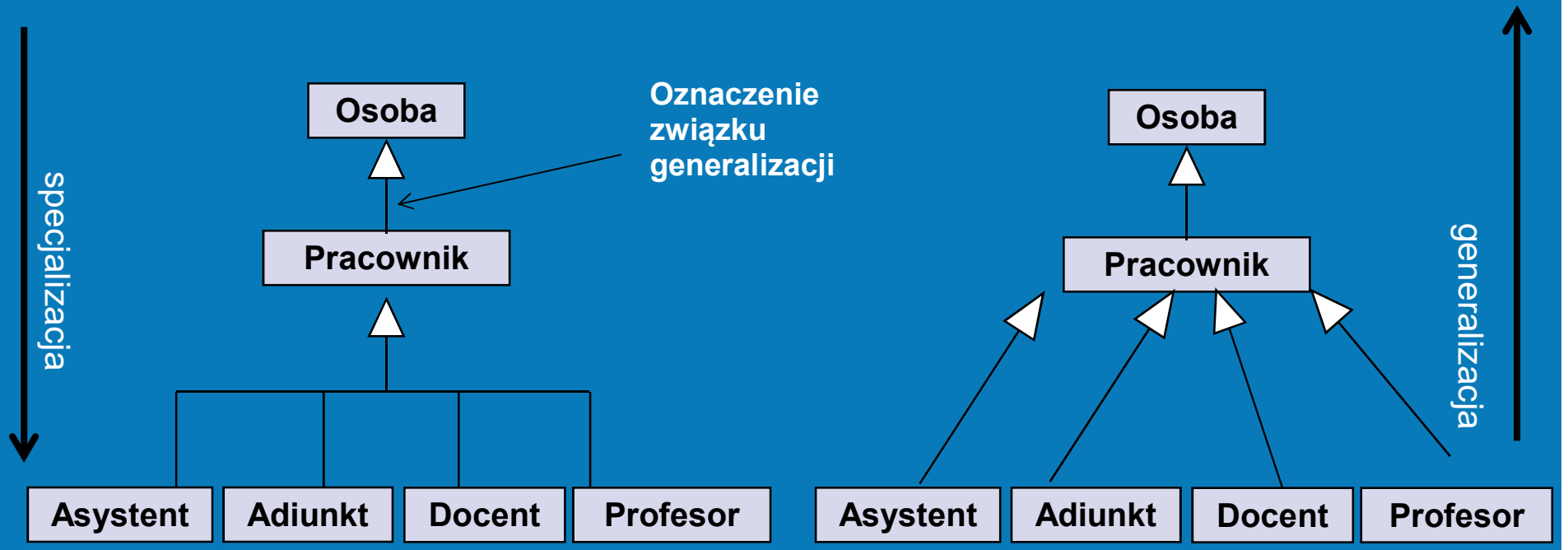
Relacja „jest rodzajem”

Dziedziczenie pozwala na tworzenie drzewa klas lub innych struktur bez pętli

Generalizacja a dziedziczenie

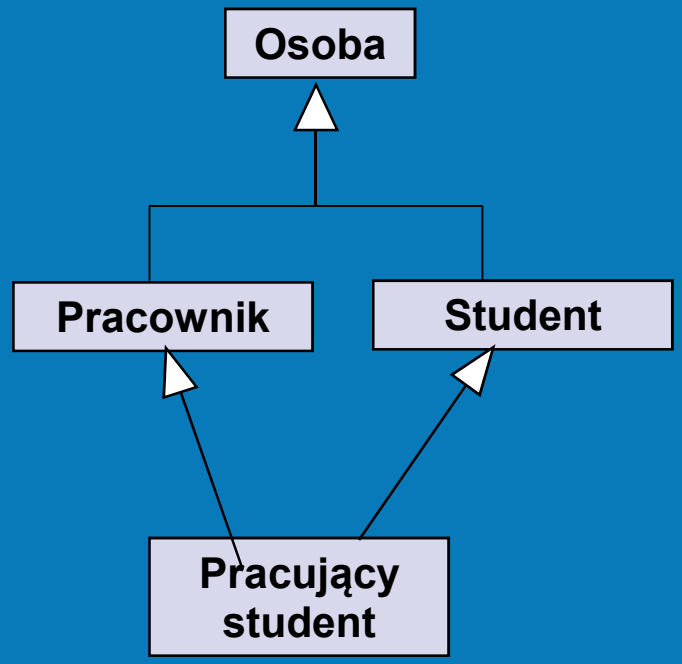
- **Generalizacja – specjalizacja (zasada zamienialności ang. Substitutability principle)**:
  - Związek, w którym obiekty specjalizowane (potomne) mogą być zamiennie stosowane z miejscu, gdzie wymagane są obiekty bardziej ogólne (bazowe).
  - Podklasa może być wykorzystywana wszędzie tam, gdzie jest używana nadklasa, ale nie odwrotnie.
  - Nazwa związku.
- **Dziedziczenie:**
  - Mechanizm, dzięki któremu bardziej specjalizowane elementy zawierają w sobie strukturę i zachowanie elementów bardziej ogólnych.
  - Mechanizm modelujący związek generalizacji.

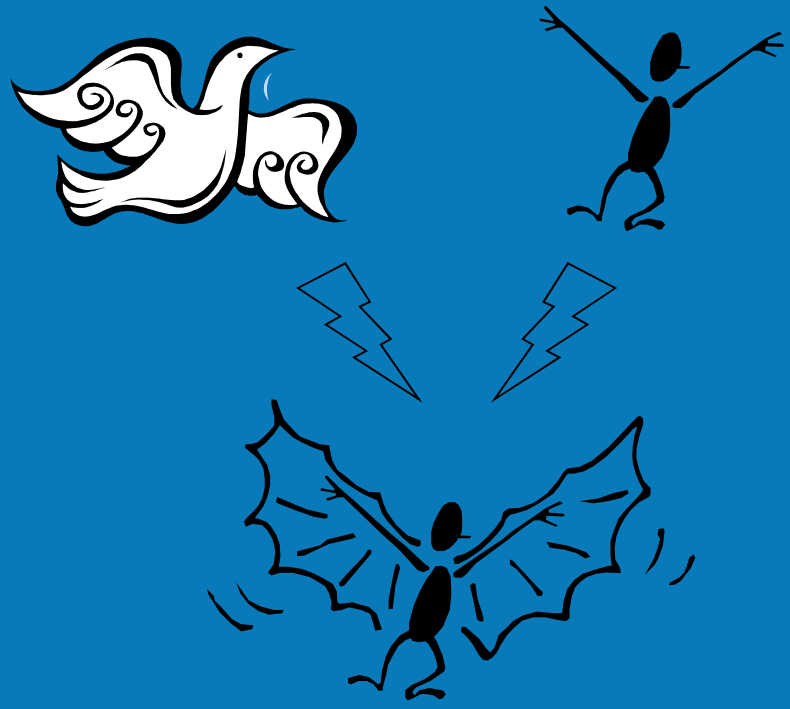
Generalizacja - specjalizacja



Dziedziczenie pojedyncze

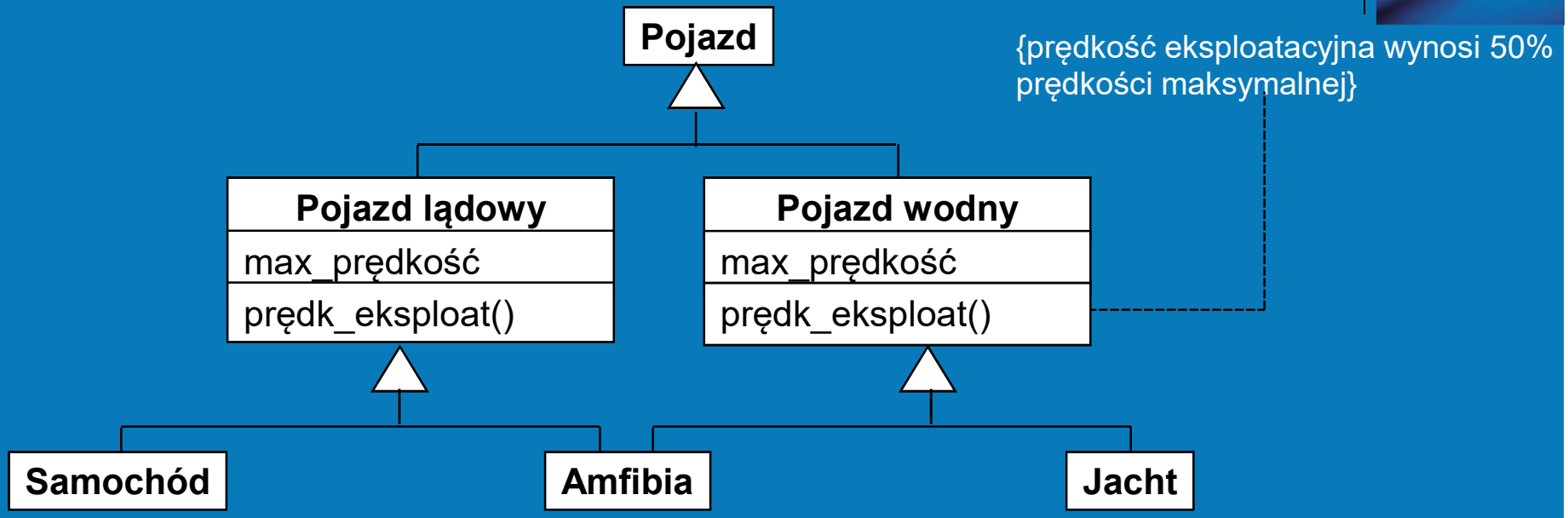
Generalizacja – specjalizacja c.d.





Dziedziczenie wielokrotne

Problemy dziedziczenia wielokrotnego



Konflikt nazw:

- Który atrybut max_prędkość ma odziedziczyć amfibia?
- Czy znaczenie metody prędk_eksploat() zależy od ścieżki dziedziczenia?
- Co z zasadą zamienialności??

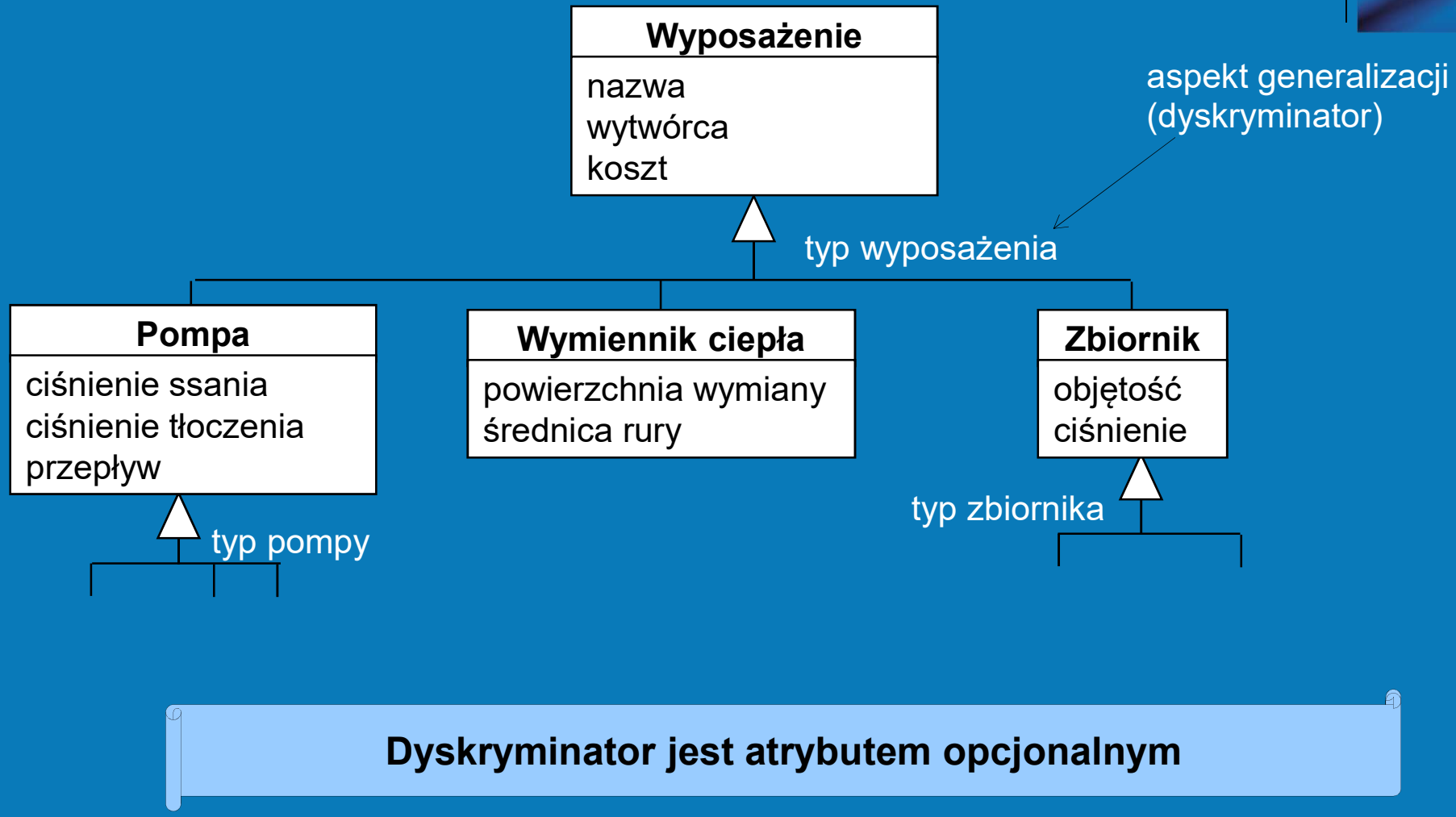
Kontrola typu:

- Jaki będzie wynikowy typ obiektów Amfibia?

Problemy dziedziczenia wielokrotnego

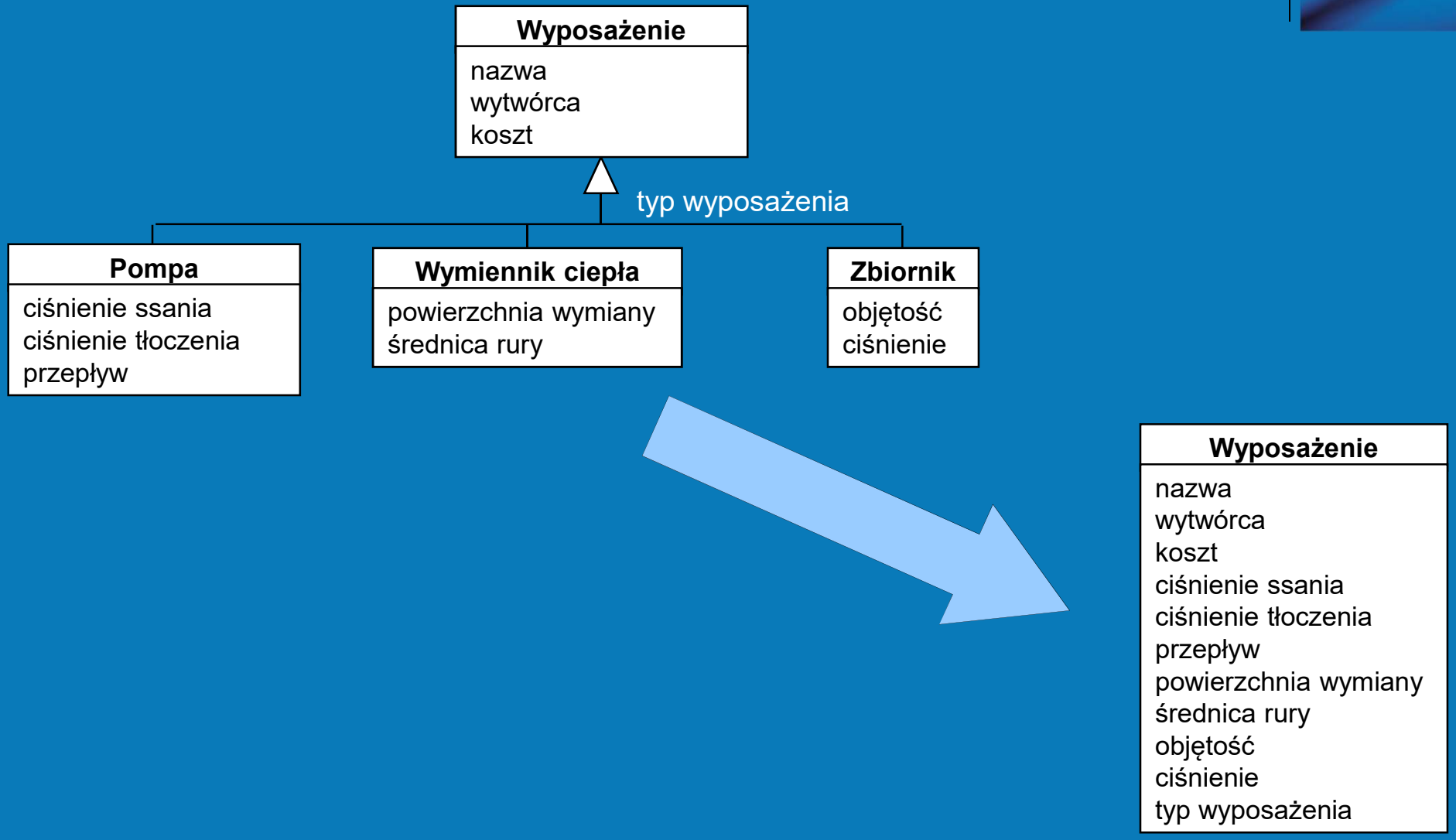
- Zazwyczaj w metodologiach obiektowych mówi się: używaj dziedziczenia wielokrotnego tylko wtedy, kiedy jest naprawdę konieczne i zawsze z ostrożnością. Jednak w języku modelowania takim jak UML dziedziczenie wielokrotne jest.
- W języku Java czy C# nie ma dziedziczenia wielokrotnego. Problem zamodelowany przez analityka przy wykorzystaniu dziedziczenia wielokrotnego, będzie musiał być zmieniony w procesie projektowania, aby zaimplementować go w jednym z tych języków. Nastąpi niewątpliwie utrata informacji z fazy analizy.

Struktury generalizacji-specjalizacji

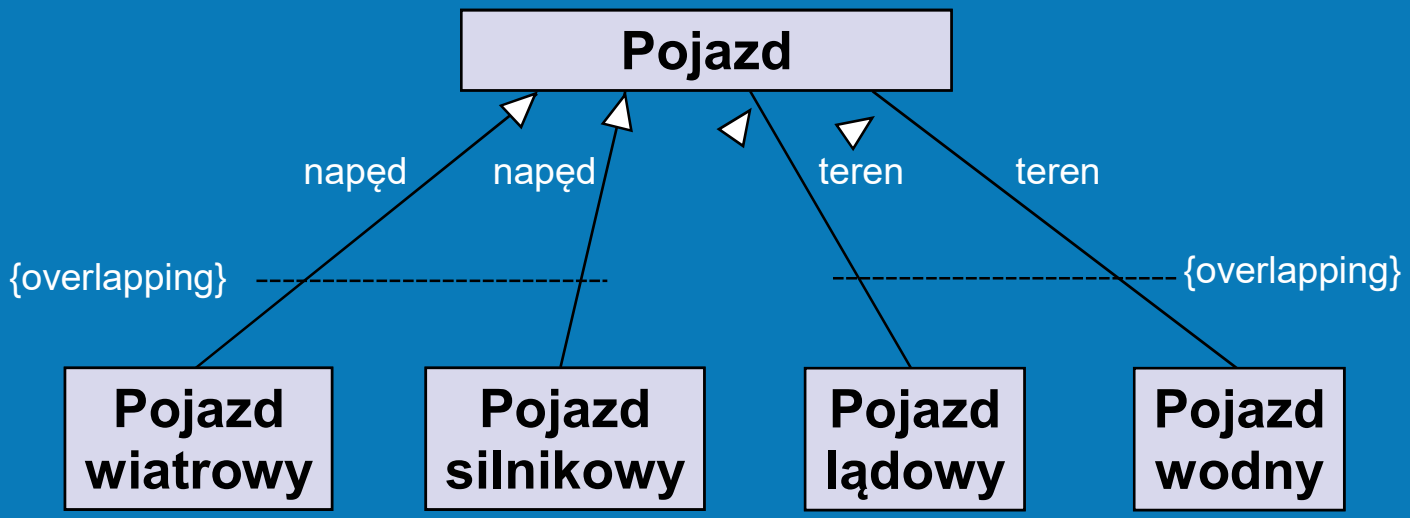


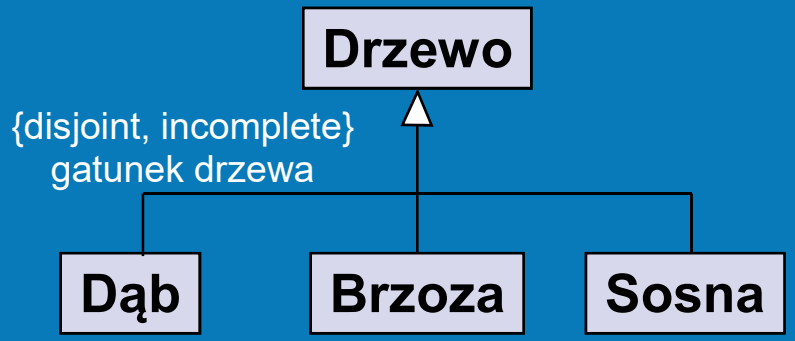
Dyskryminator jest atrybutem opcjonalnym

Struktury generalizacji-specjalizacji - spłaszczenie



Wieloaspektowe generalizacje-specjalizacje

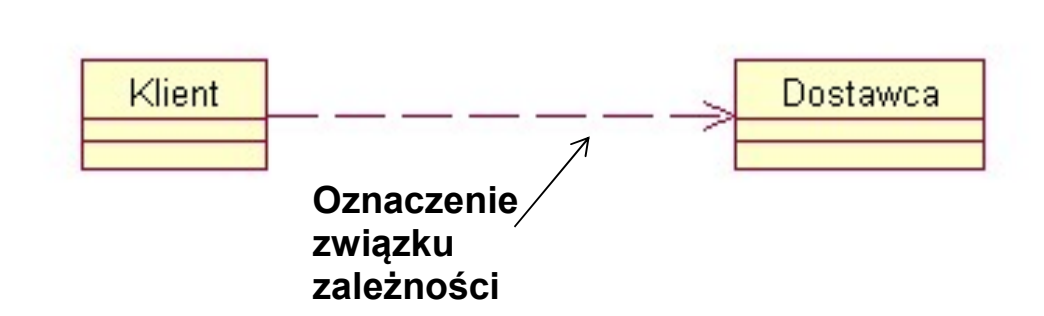




disjont - rozłączny (domyślne)
overlapping - pokrywające się
complete - zupełny (domyślne)
incomplete - niezupełny

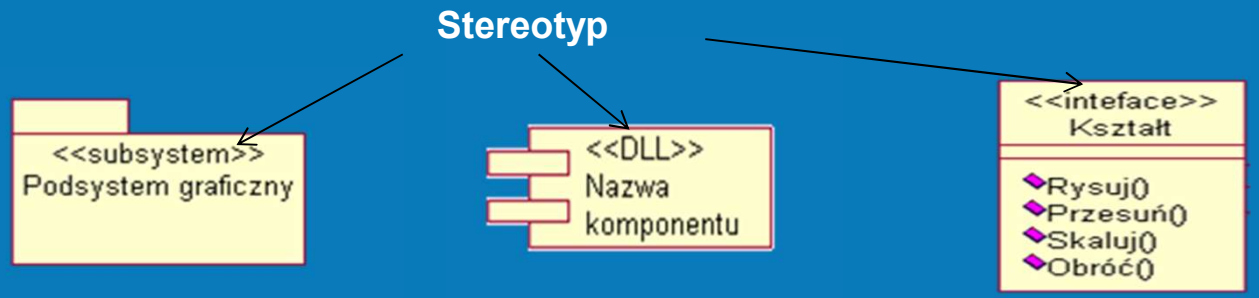
Związek zależności (ang. *dependency*)

- Związek opisujący zależność pomiędzy dwoma elementami modelu polegającą na tym, że zmiana w jednym z elementów może spowodować zmianę w drugim.
- Zależność ta może występować pomiędzy klasami, pakietami, komponentami.
- Jest to związek niestrukturalny, relacja, którą można opisać zwrotem „używa”:
  - Związek pomiędzy klientem a dostawcą, w którym klient nie ma znaczeniowej wiedzy o dostawcy, natomiast zmiana u dostawcy może wywołać zmianę u klienta; dostawca może być przedstawiany w postaci interfejsu.

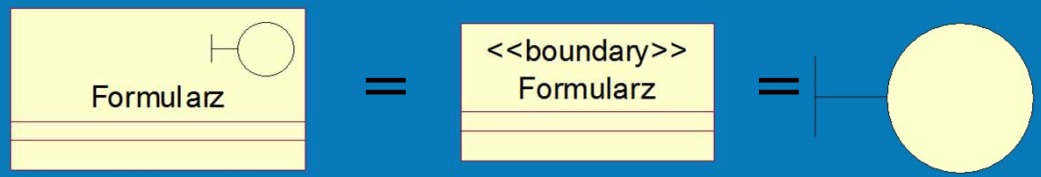


Mechanizmy rozszerzalności UML: Stereotypy

- **Stereotypy** są jednym z mechanizmów rozszerzalności UML. Dają możliwość definiowania nowych elementów, co ułatwia przystosowanie UML do specyficznego procesu projektowego, (preferencji i potrzeb konkretnego użytkownika UML) oraz pozwala na uszczegóławianie semantyki modelu



Stereotyp może być przedstawiany na różne sposoby (etykieta, dekoracja, ikona)

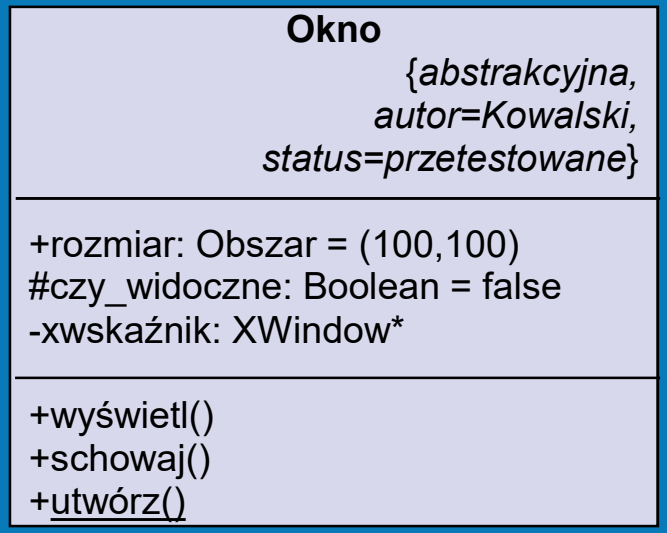


Stereotypy

- Stereotypy są wyrażeniami językowymi umożliwiającymi metaklasyfikację elementów modelu.
- Istnieje lista stereotypów dla każdego rodzaju elementów UML.
- Element modelu może mieć co najwyżej jeden stereotyp.
- Są stereotypy predefiniowane, ale użytkownicy mogą też definiować własne stereotypy.
- Stereotypy mogą mieć implikacje semantyczne (ograniczenia).

Mechanizmy rozszerzalności UML: Wartości etykietowane

- Wartość etykietowaną stanowi ciąg znaków o postaci: **słowo kluczowe = wartość**.
- Listę wartości etykietowanych (oddzielonych przecinkami) umieszcza się w {}.
- Dowolny element modelu może być skojarzony z listą wartości etykietowanych.



Wartości etykietowane

- W ten sposób można na diagramach umieścić dowolną dodatkową informację.
- Zakłada się, że narzędzia CASE umożliwią odszukanie odpowiedniego elementu na podstawie słowa kluczowego i skojarzonej z nim wartości.

Uwaga: wartości etykietowane służą raczej do opisu pojedynczego elementu modelu niż do metaklasyfikcji (patrz: stereotypy).

Mechanizmy rozszerzalności
UML: notatki

- Komentarz, który może być dodany do diagramu,
- Może dotyczyć każdego elementu UML
- W celu wskazania elementu, którego dotyczy notatka można użyć kreskowanej linii.