

Podstawy Inżynierii Oprogramowania

Unified Modeling Language -
UML

Historia UML

W latach 80-tych i na początku lat 90-tych istniało wiele notacji stosujących elementy o zbliżonej semantyce ale całkowicie różniące się sposobem reprezentacji

W 1996r „Trzej Amigos”:

G. Booch, I. Jacobson i J. Rumbaugh

zunifikowali swoje notacje tworząc jedną – UML

UML cieszy się aktualnie bardzo dużą popularnością.

Prawdopodobnie przez wiele najbliższych lat będzie dominował w obszarze analizy i projektowania.

Wady i zalety metodyk, których autorami są twórcy UML

- OMT (Rumbaugh): dobry do modelowania dziedziny przedmiotowej. Nie przykrywa dostatecznie dokładnie zarówno aspektu użytkowników systemu, jak i aspektu implementacji (konstrukcji).
- OOSE (Jacobson): dobrze podchodzi do kwestii modelowania użytkowników i cyklu życiowego systemu. Nie przykrywa dokładnie modelowania dziedziny przedmiotowej jak i aspektu implementacji (konstrukcji).
- OOAD (Booch): dobrze podchodzi do kwestii projektowania, konstrukcji i związków ze środowiskiem implementacji. Nie przykrywa dostatecznie dobrze fazy rozpoznania i analizy wymagań użytkowników.
- Celem UML jest przykrycie tych aspektów jak i również innych nie przykrytych przez w/w metodyki.

Czym jest (a czym nie jest) UML?

- Notacją graficzną
- Językiem programowania
- Metodyką
- Narzędziem

„The Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct and document the artifacts of a software-intensive system”

Konstrukcja UML

Na UML można patrzeć przez pryzmat dwóch składowych:

- **notacja** (elementy graficzne, składnia języka modelowania – ważniejsze przy modelowaniu)
- **metamodel** (ściśła semantyka poszczególnych elementów – istotne przy programowaniu graficznym i automatycznym generowaniu kodu)

Perspektywy modelowania w UML

UML jest określany jako język modelowania z 4+1 perspektywą :

- **Perspektywa przypadków użycia** – opisuje funkcjonalność systemu widzianą przez użytkowników
- **Perspektywa logiczna** – sposób realizacji funkcjonalności, struktura systemu widziana przez projektanta
- **Perspektywa implementacyjna** – zawiera moduły i interfejsy, przeznaczona dla programisty
- **Perspektywa procesowa** – podział systemu na czynności i jednostki wykonawcze (wątki, procesy, współbieżność) – służy głównie programistom i instalatorom
- **Perspektywa wdrożeniowa** – fizyczny podział elementów systemu i ich rozmieszczenie w infrastrukturze, ważna dla instalatorów

Model pojęciowy UML

- Podstawowe bloki konstrukcyjne
 - elementy
 - związki
 - diagramy
- Reguły określające sposób łączenia tych bloków
 - bloki konstrukcyjne nie mogą być rozrzucone na chybił trafił. Jak w każdym innym języku tak w UML obowiązują reguły określające jak poprawny model ma wyglądać. Dotyczą one np. nazw, ich zasięgu, kontekstu
- Mechanizmy językowe
 - specyfikacje
 - dodatki
 - rozgraniczenia
 - rozszerzenia

Podstawowe bloki konstrukcyjne UML

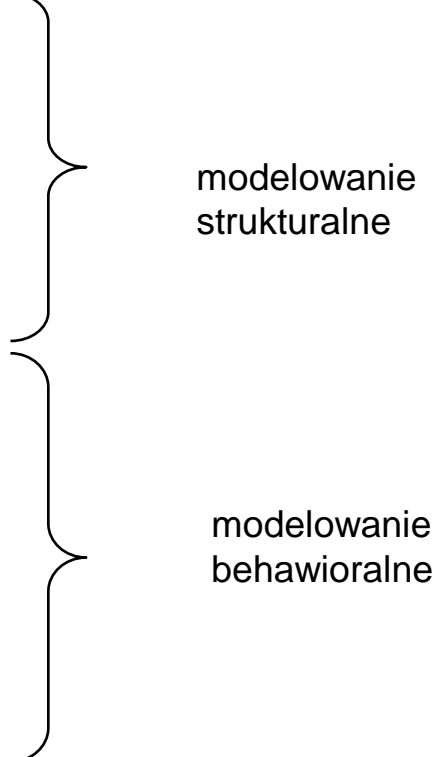
- elementy

- strukturalne (pełnią rolę rzeczowników modelu UML)
 - klasa
 - interfejs
 - kooperacja
 - przypadek użycia
 - komponent
 - węzeł
- czynnościowe (pełnią rolę czasowników modelu UML)
 - komunikat
 - maszyna stanowa
- grupujące (pełnią rolę organizacyjną, dekompozycja modelu)
 - pakiet
- komentujące (pełnią rolę objaśniającą)
 - notatka

Podstawowe bloki konstrukcyjne UML

- diagramy

Diagram – graf, którego wierzchołkami są elementy a krawędziami związki

- Diagram klas
 - Diagram obiektów
 - Diagram komponentów
 - Diagram wdrożenia
 - Diagram kooperacji
 - Diagram przypadków użycia
 - Diagram czynności
 - Diagram sekwencji
 - Diagram stanów
- 
- The diagram uses two large curly braces on the right side to group the list items. The top brace groups the first four items (Diagram klas, Diagram obiektów, Diagram komponentów, Diagram wdrożenia) and is labeled 'modelowanie strukturalne'. The bottom brace groups the remaining five items (Diagram kooperacji, Diagram przypadków użycia, Diagram czynności, Diagram sekwencji, Diagram stanów) and is labeled 'modelowanie behawioralne'.
- modelowanie strukturalne
- modelowanie behawioralne

Przykładowy diagram przypadków użycia

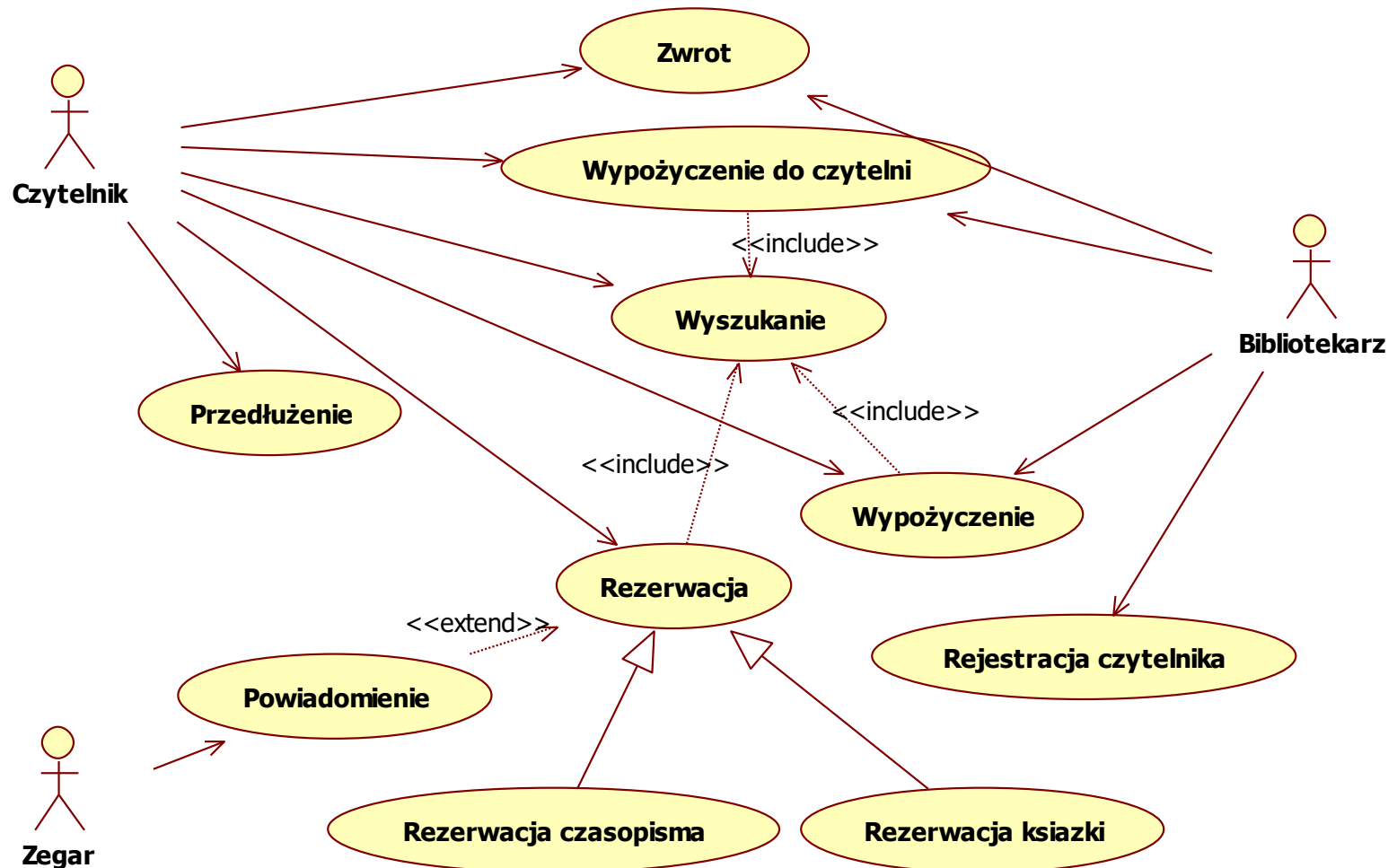


Diagram klas

- Jest podstawowym diagramem struktury logicznej systemu
- Przedstawia podział odpowiedzialności systemu pomiędzy jego klasy oraz wymienianych pomiędzy nimi komunikatów
- Zawiera największą ilość informacji i stosuje największą liczbę symboli
- Najczęściej używany diagram UML (także do generowania kodu na podstawie modelu)

Diagram klas – klasa

Okno

Okno
rozmiar
czy_widoczne

Okno
rozmiar
czy_widoczne
wyświetl()
schowaj()

Okno
rozmiar: Obszar
czy_widoczne: Boolean
wyświetl()
schowaj()

Diagram klas- atrybuty

widoczność nazwa : typ [krotność] {ograniczenia} = wartość domyślna

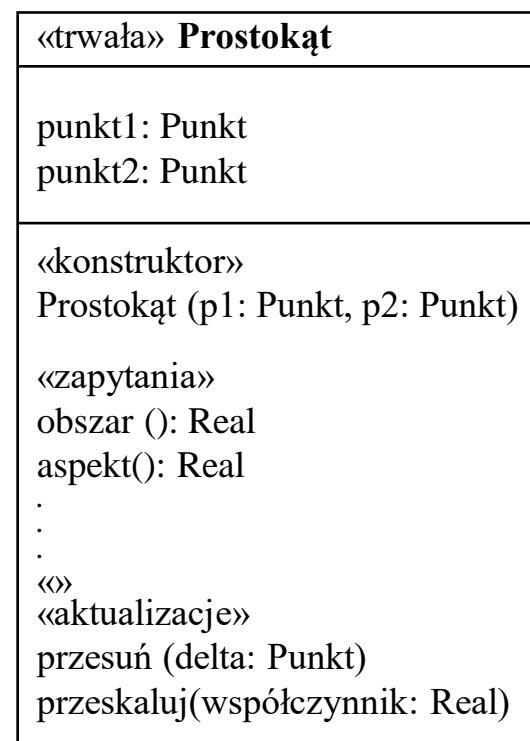
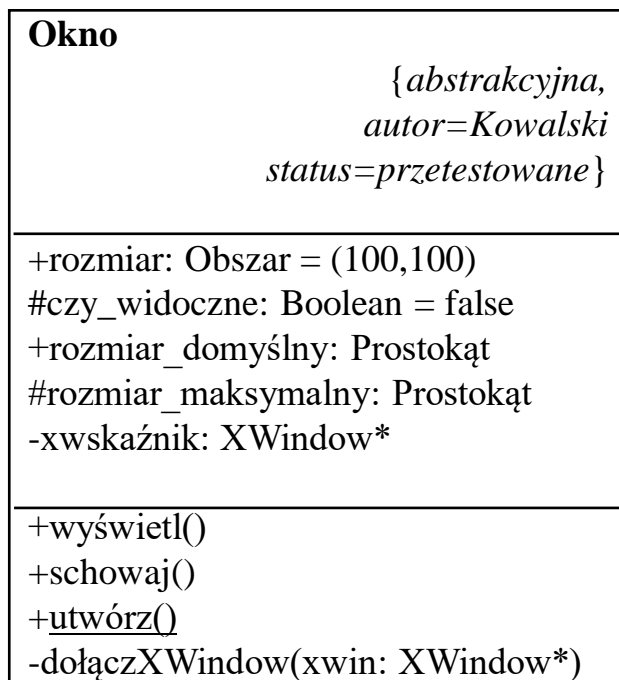
- Widoczność
 - + publiczny (widoczny z każdego miejsca w systemie)
 - # chroniony (widoczny we własnej klasie i jej podklasach)
 - - prywatny (widoczny tylko we własnej klasie)
 - ~ publiczny wewnątrz pakietu
- Krotność (liczba obiektów jakie można powiązać z daną cechą)
 - dolna granica .. górna granica (1; 0..1; 1..*; 1,3,5; *)
- Ograniczenia np.:
 - {ordered} {unordered}
 - {unique} {nonunique}
 - {readOnly}
 - {frozen}

Diagram klas- operacje

widoczność nazwa (par1,par2,...): typ {ograniczenia}

- Parametry opisywane są tak jak atrybuty poprzedzone informacją o kierunku przekazania (in, out)
- Ograniczenia np.:
 - {query} operacja nie modyfikuje stanu obiektu – jest zapytaniem
 - <<exception>> metoda może zgłaszać wyjątek

Diagram klas – przykłady klas



Stereotypy zostały tu użyte do metaklasyfikacji metod.

Diagram klas - związki

- **Zależność** – najprostszy i najśłabszy rodzaj relacji (zmiana jednej klasy wpływa na drugą)
 - <<call>> operacje w klasie A wywołują operacje w klasie B
 - <<create>> klasa A tworzy instancje klasy B
 - <<instantiate>> obiekt A jest instancją klasy B
 - <<use>> do zaimplementowania klasy A wymagana jest klasa B

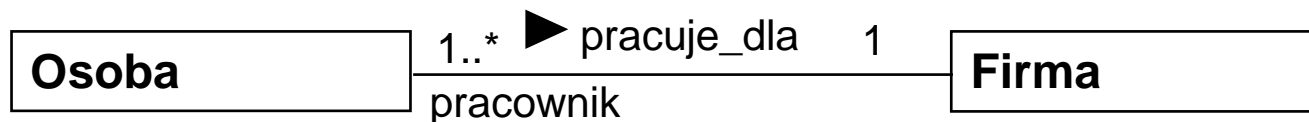
<-----

Diagram klas - związki

- **Asocjacja** – silniejsza relacja niż zależność, reprezentuje czasowe powiązanie między obiektami dwóch klas, ale czas życia obu obiektów nie jest od siebie zależny – usunięcie jednego z obiektów nie powoduje usunięcia drugiego (jest używana jako alternatywny obok atrybutu sposób zapisu cech klasy – zwykle obiekty proste modelowane są jako atrybuty a obiekty dostępne przez referencję jako asocjacje)
 - nawigowalność asocjacji określa wiedzę o sobie nawzajem obiektów uczestniczących w relacji

Diagram klas – oznaczanie asocjacji

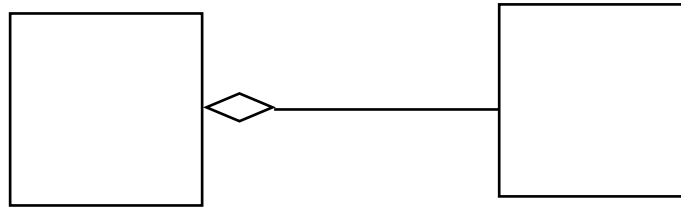
- **Czarny trójkąt** określa kierunek (czytania) wyznaczony przez nazwę asocjacji. W danym przypadku określa on, że osoba pracuje dla firmy, a nie firma pracuje dla osoby. Nazwy asocjacji, takie jak np. *pracuje_dla*, wyznaczają znaczenie tej asocjacji w modelu pojęciowym opisującym dziedzinę przedmiotową.



- 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.

Diagram klas - związki

- **Agregacja** – silniejsza forma asocjacji (istnieje właściciel i obiekt podrzędny powiązane czasem swojego życia właściciel nie jest jednak wyłącznym właścicielem obiektu podrzędnego)



- **Kompozycja** – najsilniejszy związek łączący klasy (relacja całość-część w której części są tworzone i zarządzane przez obiekt będący całością)

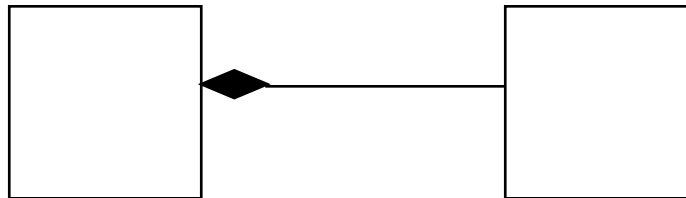


Diagram klas - Liczność asocjacji

Jeżeli asocjacja wiąże klasy *A* i *B*, to istotne jest:

- jaka jest minimalna liczba obiektów *B* powiązana z jednym obiektem *A*, $A \rightarrow B$
- jaka jest maksymalna liczba obiektów *B* powiązana z jednym obiektem *A*, $A \rightarrow B$
- jaka jest minimalna liczba obiektów *A* powiązana z jednym obiektem *B*, $B \rightarrow A$
- jaka jest maksymalna liczba obiektów *A* powiązana z jednym obiektem *B*, $B \rightarrow A$.

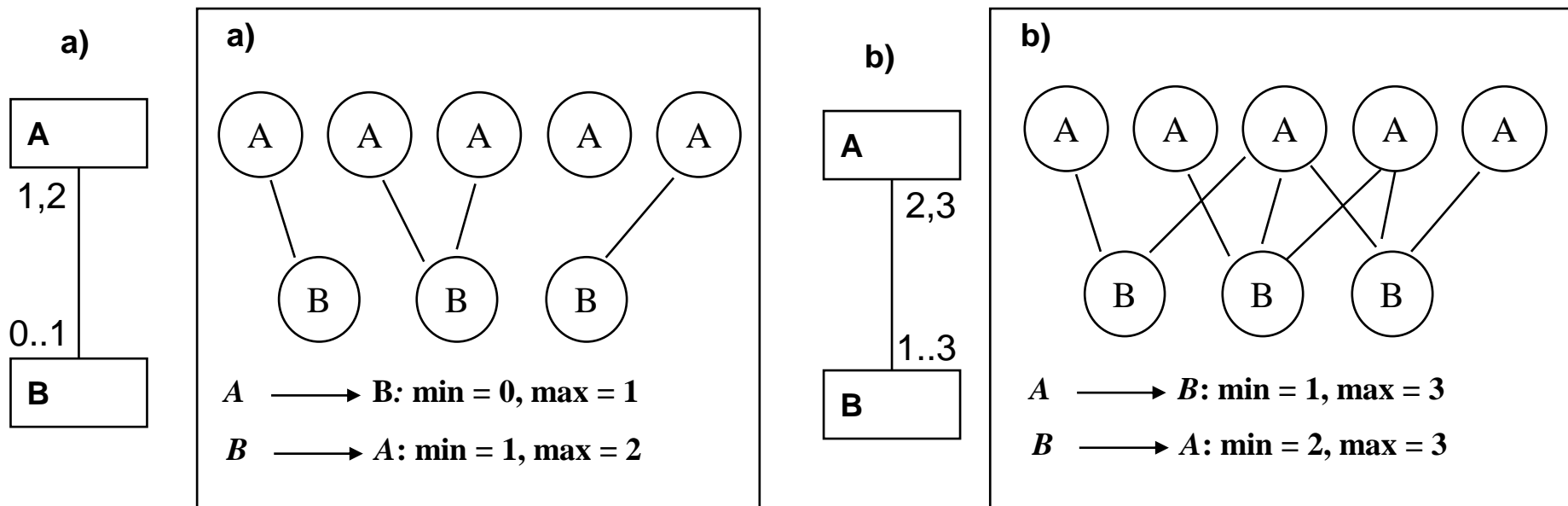


Diagram klas - przykład

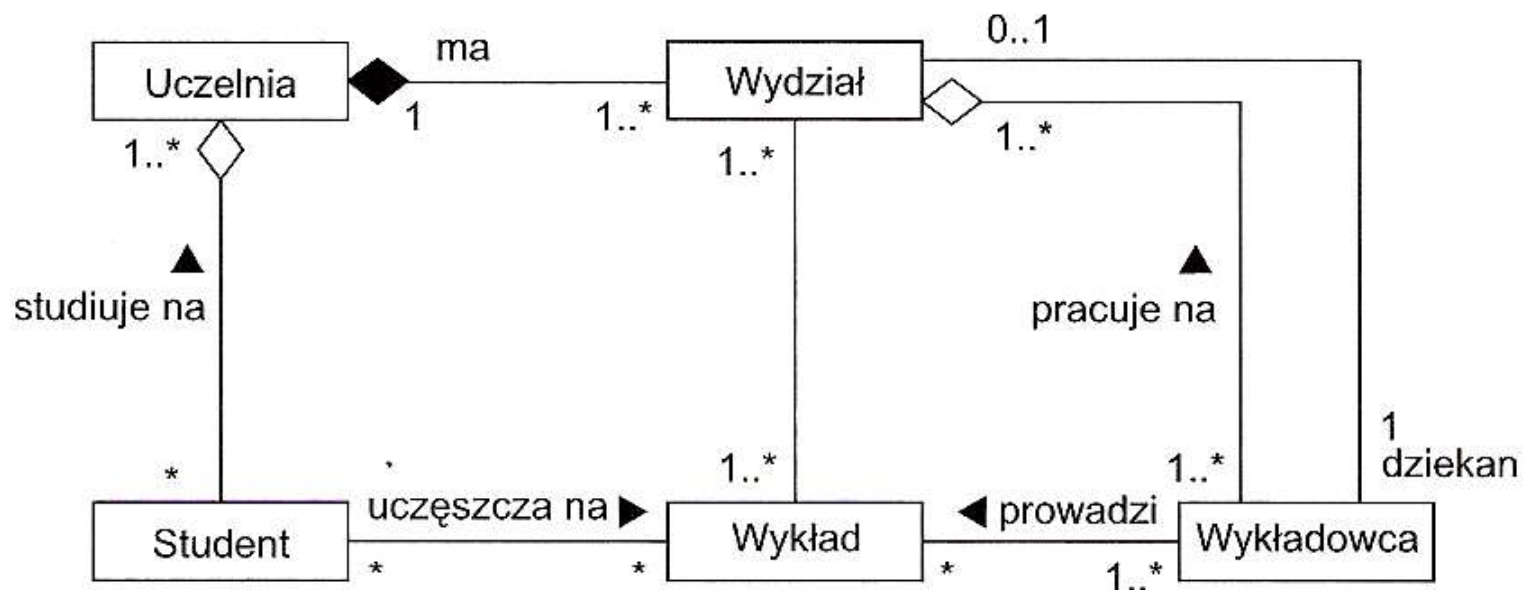
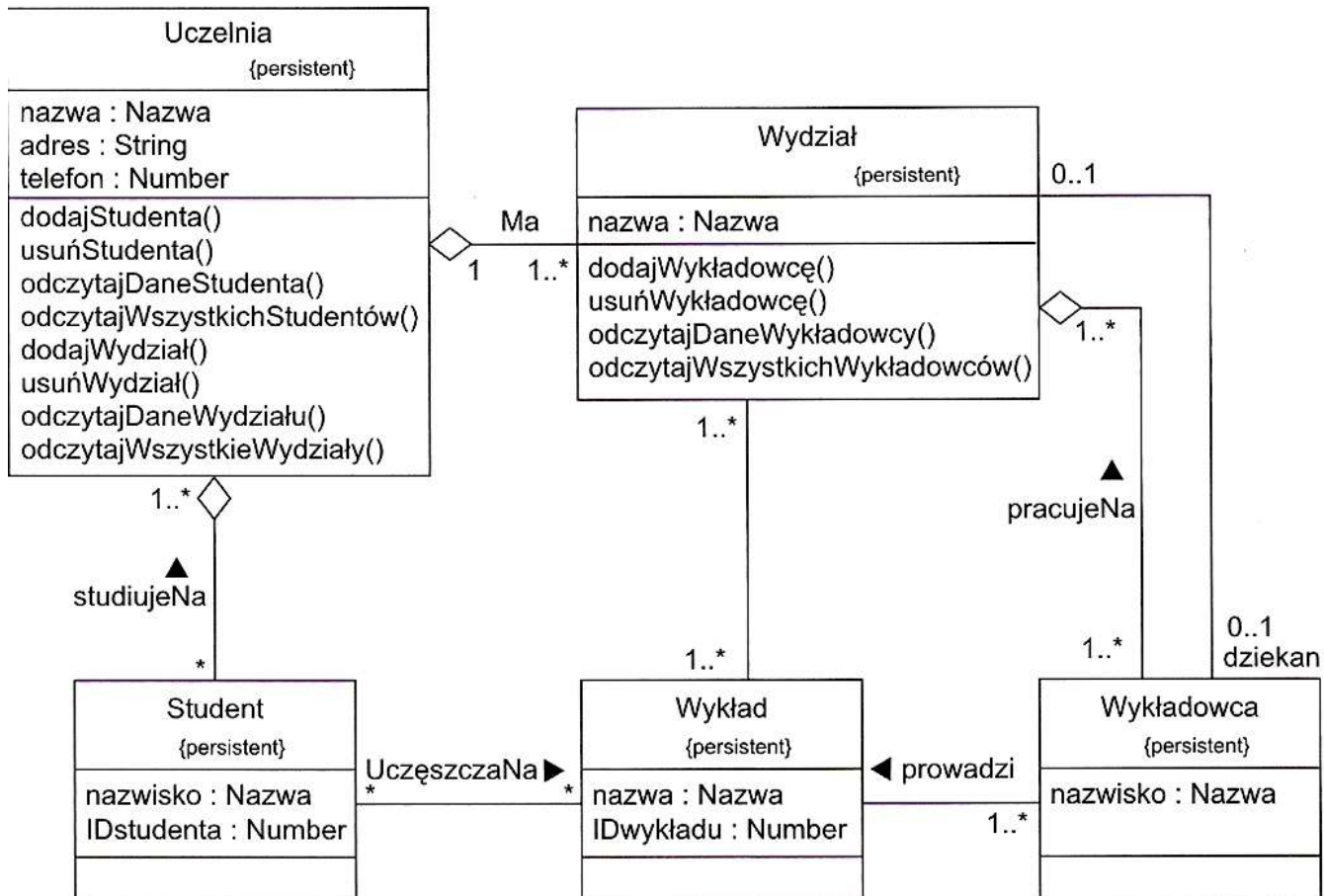
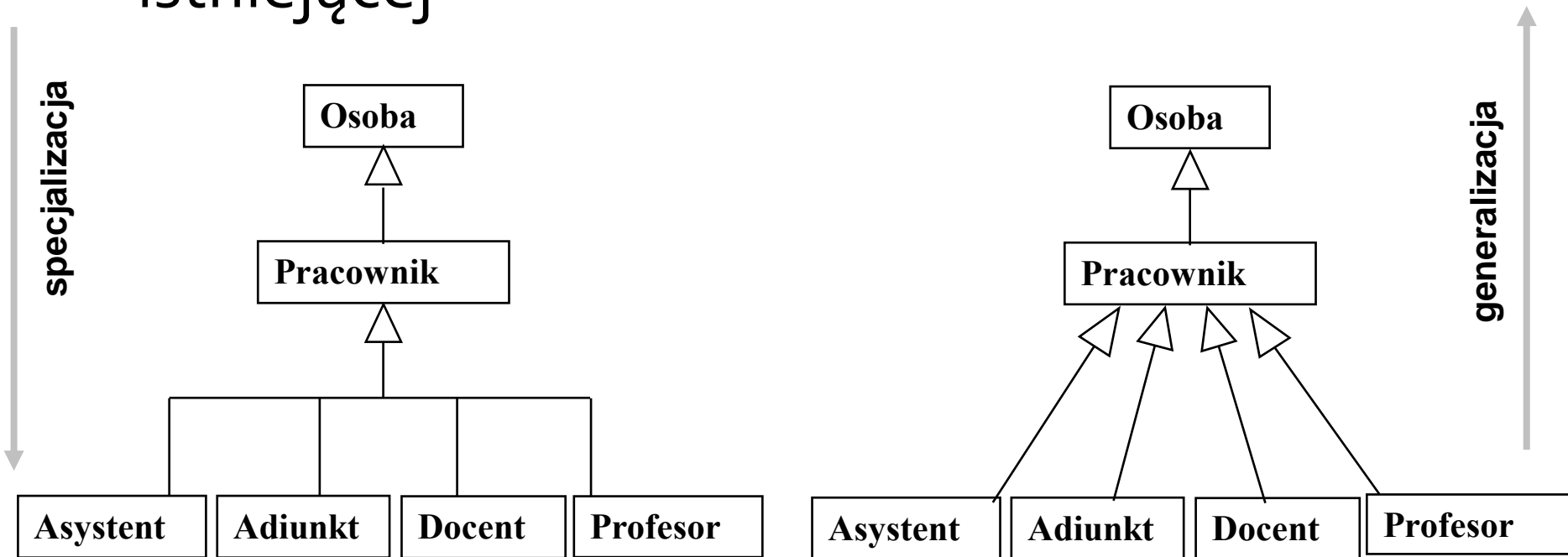


Diagram klas - przykład



Diagramy klas - dziedziczenie

- Definiowanie nowej klasy na podstawie klasy istniejącej



Tworzymy diagram klas – przykładowa specyfikacja

Hurtownia posiada produkty różnego rodzaju, każdy z nich jest identyfikowany przez unikatową nazwę handlową. Produkt jest opisany ceną jednostkową, wagą jednostkową oraz datą ważności dla produktów krótkoterminowych. Hurtownia przechowuje dane dostawców: nazwa, adres, telefon. Klient kontaktuje się z hurtownią wysyłając zamówienia. Jedno zamówienie może dotyczyć wielu produktów. Hurtownia posiada samochody transportowe o różnej ładowności.

Diagram klas - hurtownia

Visual Paradigm for UML Standard Edition (Technical University Of Lodz)

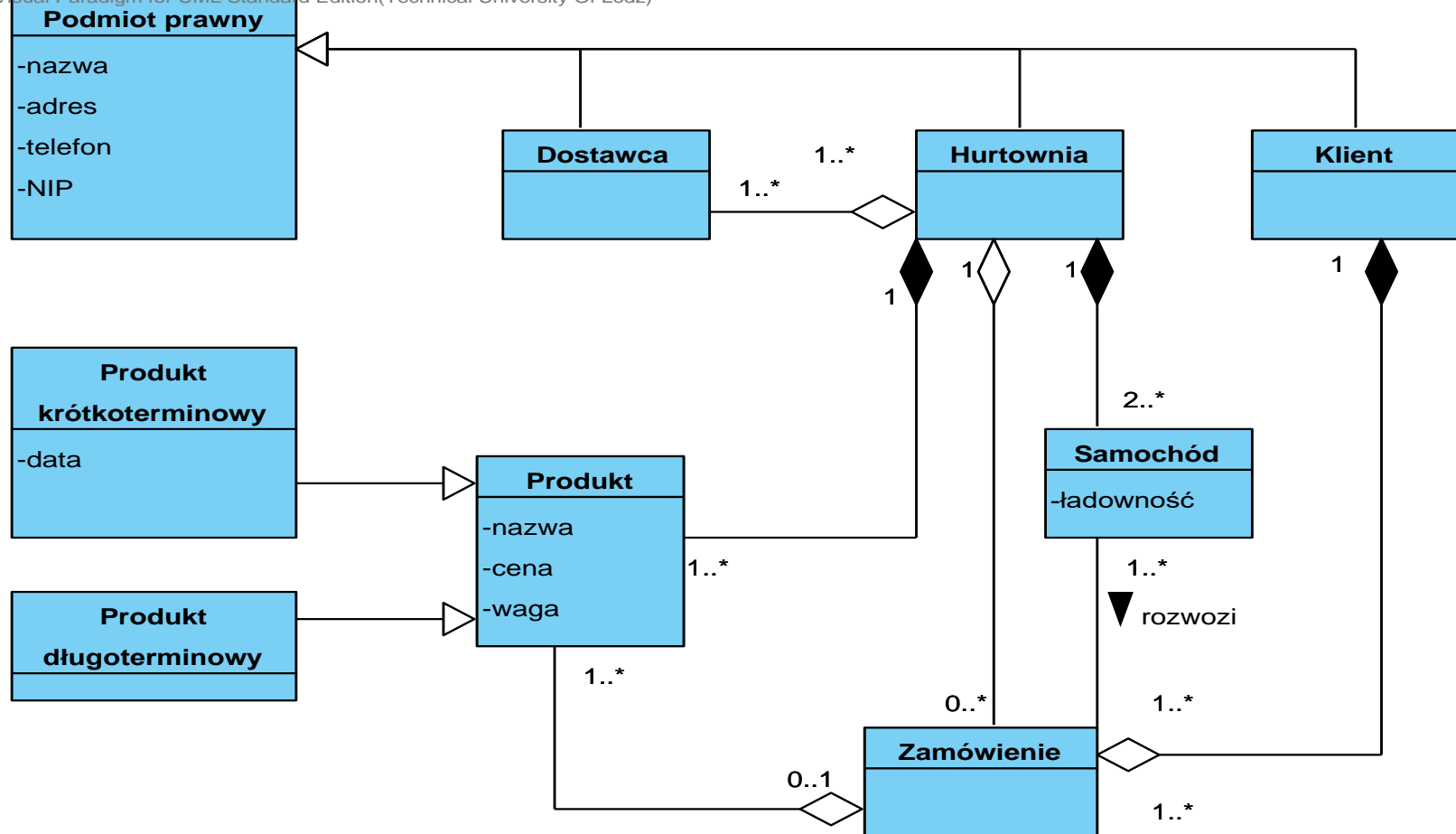


Diagram klas – typowe błędy

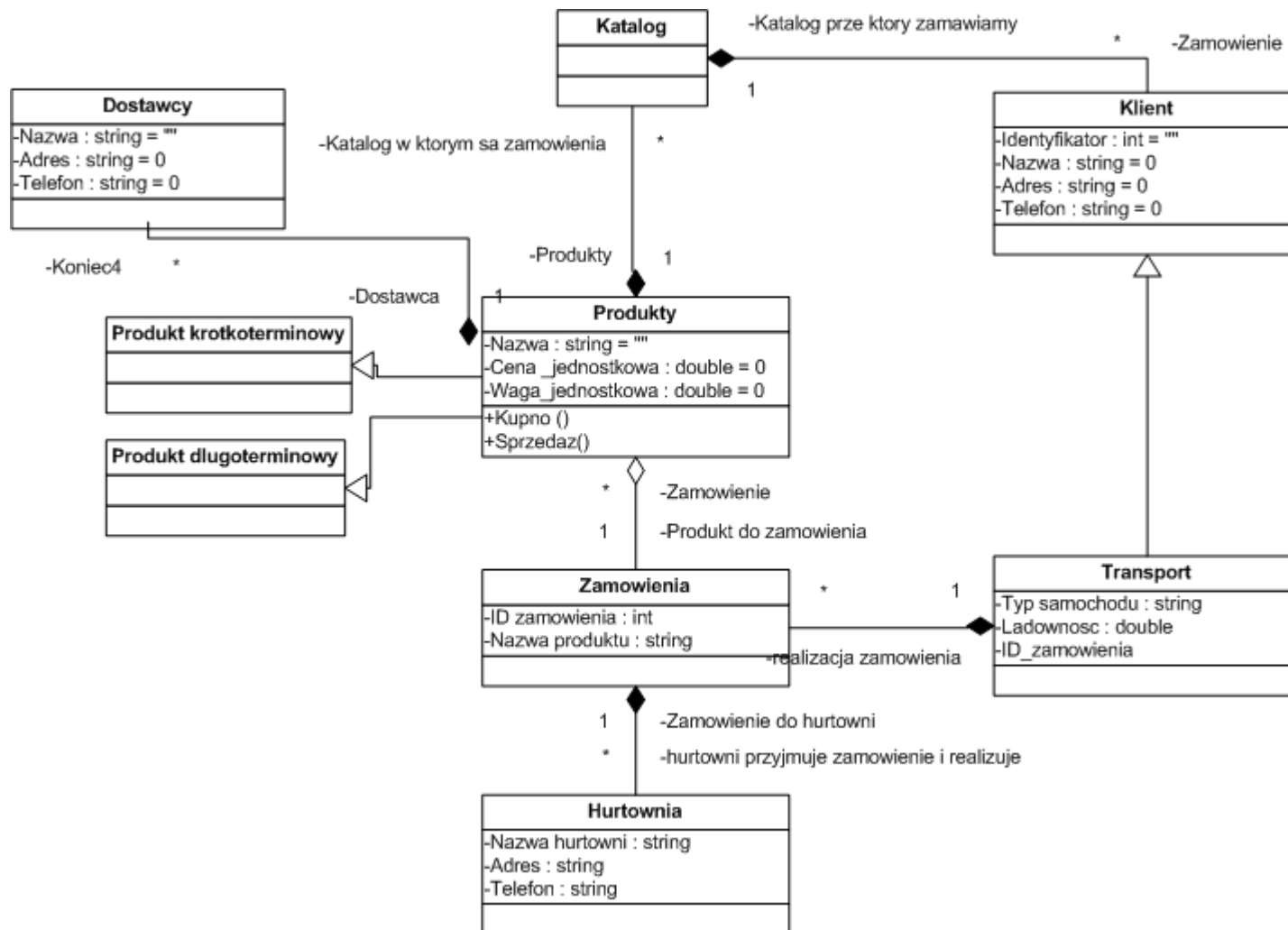


Diagram klas

– klasa abstrakcyjna, interfejs

- **Klasa abstrakcyjna** – deklaruje wspólną funkcjonalność grupy klas. Nie może posiadać obiektów i musi definiować podklasy
- **Interfejs** – deklaruje grupę operacji bez podawania ich implementacji
- Klasa abstrakcyjna może posiadać implementację niektórych operacji natomiast interfejs jest czysto abstrakcyjny
- Przyjętym sposobem oznaczania klas abstrakcyjnych jest zapisywanie ich pochyłą czcionką

Diagram klas

– klasa abstrakcyjna, interfejs

Visual Paradigm for UML Standard Edition (Technical University Of Lodz)

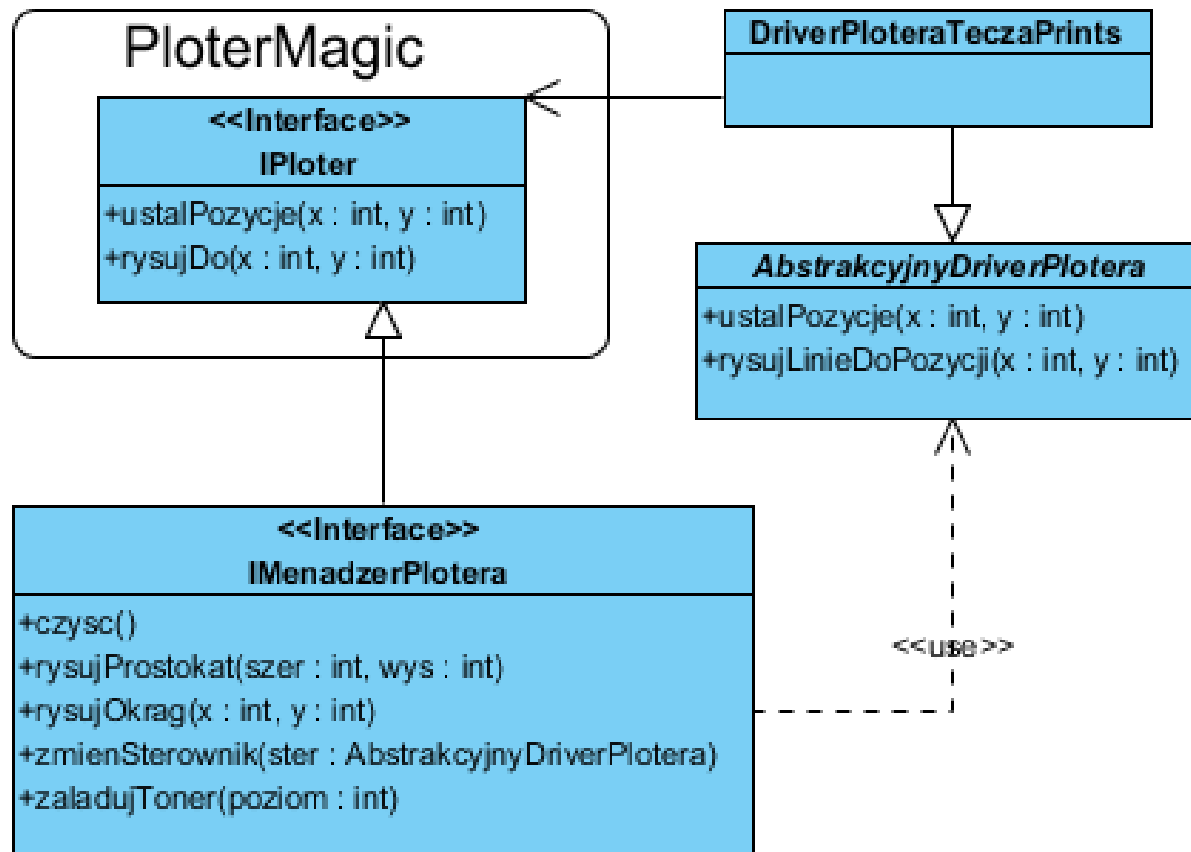


Diagram obiektów

- Prezentuje możliwą konfigurację obiektów w określonym momencie – jest instancją diagramu klas
- Posługuje się identycznymi symbolami co diagram klas, ale nazwy instancji są **podkreślone** (nazwa składa się nazwy obiektu i nazwy klasy oddzielonych dwukropkiem)
- Przydają się w przypadku szczególnie skomplikowanych zależności, trudnych do przedstawienia na diagramie klas

Diagram obiektów - przykład

