Model-Driven Engineering

Radosław Klimek

2015-22

http://home.agh.edu.pl/rklimek

Diagramy klas i obiektów

Informacje podstawowe

- Należą do najpopularniejszych diagramów.
- Klasy są rdzeniem każdego systemu obiektowego a struktura systemu składa się ze zbioru elementów nazywanych obiektami.
- Diagramy klasy prezentują obiekty oraz zależności między nimi.
- Zawierają informacje o statycznych związkach między elementami (klasami), pokazują typy obiektów istniejących w systemie.
- Pokazują także typy enumeratywne, interface'y oraz inne.
- Klasy są ściśle powiązane z technikami programowania zorientowanego obiektowego.
- Są jednymi z istotniejszych diagramów w UML.

Definicje

Klasa (*class*) to opis zbioru obiektów, które mają takie same atrybuty, związki i znaczenie.

Obiekt *(object*) – konkretne wystąpienie abstrakcji; byt o dobrze określonych granicach i tożsamości, obejmujący stan i zachowanie; egzemplarz klasy.

• Każda klasa musi mieć przypisaną nazwę prostą (rzeczownik) lub ścieżkową (poprzedzoną nazwą pakietu).
• Symbolem jest prostokąt, podzielony na trzy sekcje:
- nazwy
- atrybutów
- operacji
• Można uzupełnić o dodatkowe sekcje (np. wyjątków)

Diagramy klas

- Zawiera informację o statycznych związkach między elementami (klasami),
- Klasy są ściśle powiązane z technikami programowania zorientowanego obiektowego,
- Są jednymi z istotniejszych diagramów w UML.

Nazwy proste i ścieżkowe

Sensor

Klient

Ściana

RegułyPrzedsiębiorstwa:: WykrywaczOszustw

Perspektywy przy rysowaniu diagramów klas

- Pojęciowa
- Specyfikacyjna
- Implementacyjna

Perspektywa pojęciowa

W tej perspektywie rysuje się diagramy reprezentujące pojęcia analizowanej dziedziny, które są związane z klasami je implementującymi. Model pojęciowy powinien być tworzony niezależnie od oprogramowania, które może implementować ten model.

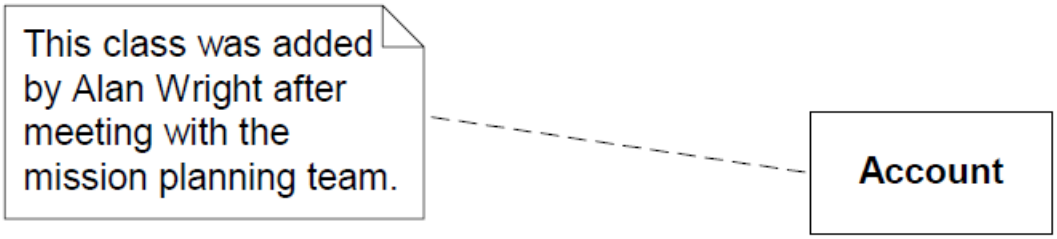
Perspektywa specyfikacyjna

Dotyczy interfejsów a nie już samej implementacji.

Kładzie duży nacisk na odróżnienie między interfejsem a implementacją. Jest to ważna perspektywa ponieważ kluczem do efektywnego programowania obiektowego jest programowanie sterowane interfejsem klasy.

Komentarze

Komentarz to adnotacja tekstowa, którą można dołączyć do zestawu elementów. Komentarz daje możliwość dołączania różnych uwag do elementów. Komentarz nie ma żadnej siły semantycznej, ale może zawierać informacje przydatne dla projektanta.



Problematyka widoczności

Za pomocą cechy widoczności klasy można kontrolować dostęp do atrybutów, operacji a nawet całych klas.

Rodzaje widoczności:
- Publiczny,
- Chroniony,
- Pakietowy,
- Prywatny.

Poziom publiczny +

Poziom określany za pomocą symbolu (+) użytego przed skojarzonym atrybutem lub operacją. Ten poziom widoczności udostępnia bezpośredni atrybuty i operacje dla dowolnej innej klasy.

Interfejsem publicznym nazywamy zbiór operacji i atrybutów zadeklarowanych jako publiczne w danej klasie.

Dobrą praktyką jest unikanie stosowanie atrybutów publicznych. Jednak można udostępnić dany atrybut innym klasom jeżeli jest on stały to oznacza że posiada początkową niezmienną wartość, ma nadaną właściwość typu *readonly.* Ten sposób udostępnienia atrybutów jest bezpieczny, ponieważ jego wartość nie może zostać zmieniona.

Poziom chroniony

Atrybuty i operacje chronione są określane przy użyciu symbolu (#). Elementy zadeklarowane jako chronione mogą być używane przez metody będące częścią danej klasy, jak również przez metody dowolnej innej, która ją dziedziczy. Elementy chronione nie mogą być używane przez żadną klasę niewywodzącą się z tej, w której są one zdefiniowane.

Poziom pakietowy ~

Oznaczany jest przy użyciu znaku (~). Jeżeli klasa posiada atrybut bądź operacje zadeklarowaną jako poziom widoczności pakietowy to wtedy bezpośredni dostęp do tego mają wszystkie klasy w tym samym pakiecie. Klasy spoza pakietu nawet jeżeli dziedziczą po tej klasie nie mają dostępu do atrybutów i operacji.

Poziom prywatny -

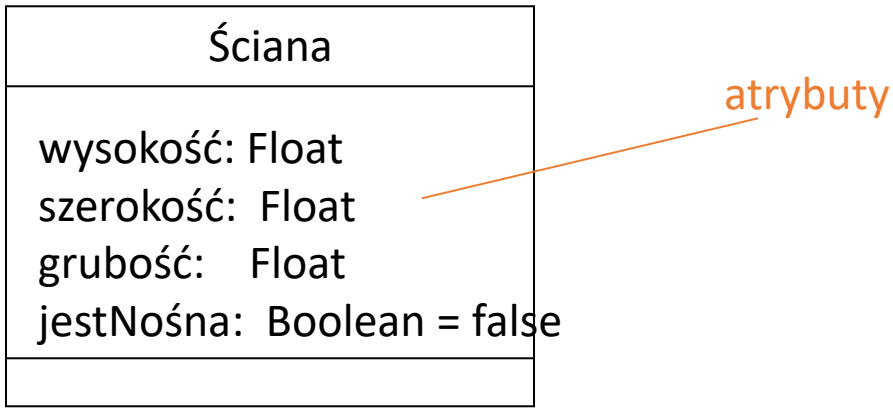
Oznaczany jest przy użyciu symbolu (-). Jest on najbardziej ograniczonym rodzajem klasyfikacji widoczności. Jedynie klasa, która zawiera element oznaczony jako prywatny, może mieć dostęp do danych umieszczonych w prywatnym atrybucie lub też wywołać prywatną operację.

Stosujemy gdy chcemy atrybut lub operacja nie zależały od niej żadna inna część systemu. Przydatne jest to w sytuacji gdy będziemy zmieniać ten atrybut lub operację i nie chcielibyśmy, aby zmianie uległa inna klasa używająca tego elementu.

Atrybuty klas

Atrybut jest nazwaną właściwością (cechą) klasy. Określa zbiór wartości, jakie można przypisać do poszczególnych egzemplarzy tej klasy.

Klasa może mieć dowolną liczbę atrybutów, lub nie mieć wcale. Atrybut reprezentuje właściwość modelowanego bytu, określoną dla wszystkich jego wystąpień.



Deklaracja atrybutu

[widoczność] nazwa [liczebność] [:typ] [=wartość początkowa] [właściwość]

Atrybut posiada sygnaturę określającą właściwości jego poziomu widoczności, nazwę oraz typ. Nazwa atrybutu jest jedyną częścią sygnatury, która musi być w niej bezwzględnie obecna, aby klasa była poprawna.

Nazwa atrybutu może być dowolnym zestawem znaków, jednakże żadne dwa atrybuty w tej samej klasie nie mogą mieć tej samej nazwy.

Typ atrybutu może być różny w zależności od sposobu implementacji klasy w systemie.

Modelowanie atrybutów

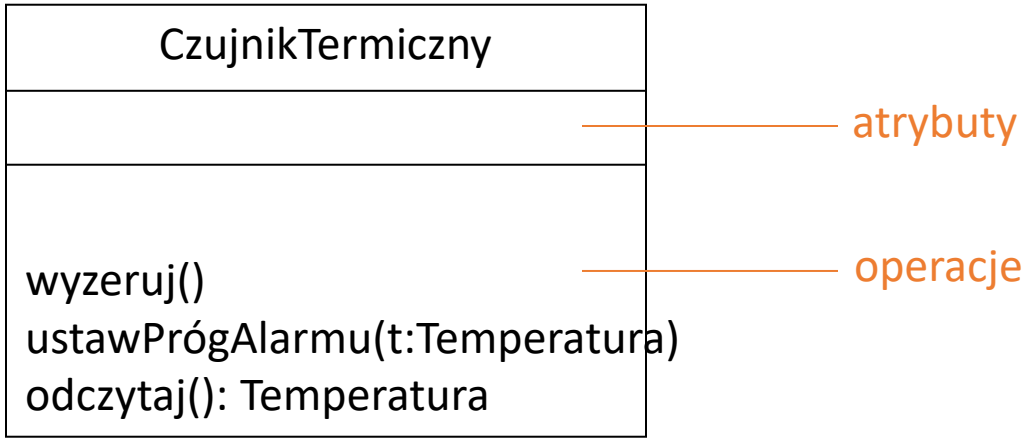
Podstawowym celem modelowania systemu jest przekazanie projektu innym osobom. Atrybuty powinny mieć tak dobraną nazwę, która będzie opisywać reprezentowaną przez niego informację. Można też pamiętać o tym że nazwa powinna spełniać konwencje nazewniczą w zależności od implementacji w konkretnym języku programowania.

Ponadto:
atrybutom można nadawać różne właściwości, takie jak: union, subset, redfine, composite oraz readOnly. Ta ostatnia z nich jest najpopularniejsza i oznacza że wartość początkowa nie może być Zmieniana.

Operacje klas

Operacja to implementacja pewnej usługi, której wykonania można zażądać od każdego obiektu klasy.

Klasa może mieć dowolną (≥ 0) liczbę operacji.



Deklaracja operacji

[widoczność] nazwa [(lista_parametrów)] [: typ_wyniku] [właściwości]

gdzie lista parametrów:
[tryb] nazwa : typ [=wartość_domyślna]

Są używane do określenia informacji udostępnianej operacji w celu wykonania przez nią zadania. Do operacji może być przekazany więcej niż jeden parametr, co jest możliwe dzięki oddzieleniu parametrów przy użyciu przecinka.

Tryb:
- in – wyjściowy, nie może być modyfikowany
- out – wyjściowy, może być modyfikowany
- Inout – wejściowy, wyjściowy, może być modyfikowany

Właściwości operacji

leaf – funkcja niepolimofriczna (nie może być nadpisana)
isQuerry – funkcja nie zmieniająca obiektu
sequental, guarded, concurent – mają zastosowanie przy operacjach współbieżnych

Okres istnienia klas składowych

Można podzielić na dwa typy:
- statyczne,
- niestatyczne

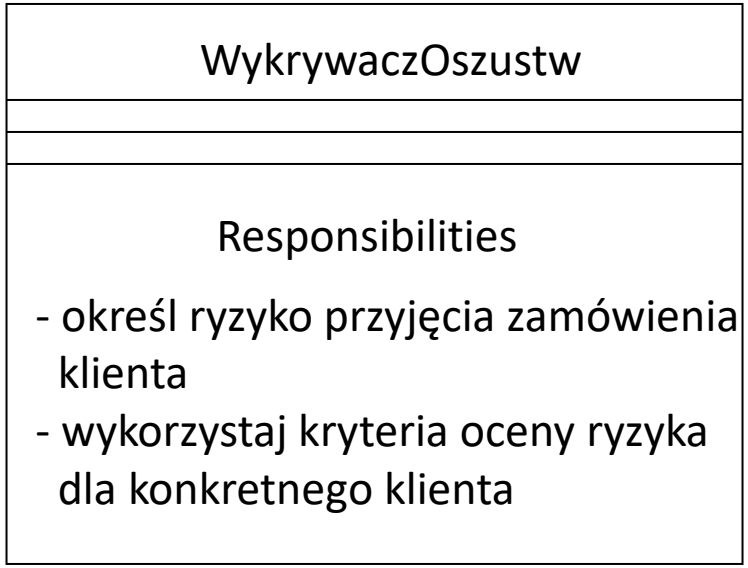
Jeżeli jakaś operacja bądź atrybut jest **niestatyczny** oznacza to, że jest skojarzony z instancjami lub inaczej obiektami klasy. Oznacza to że każdy obiekt danej klasy będzie dysponować swoją własną kopią atrybutów oraz operacji.

W przypadku jeżeli atrybut lub operacja jest **statyczna**, to wszystkie obiekty danej klasy będą współdzielić tę samą kopię operacji lub atrybutu. Czas istnienia jest dłuższy niż dowolny obiekt stworzony na podstawie klasy. Pola statyczne są przydatne w momencie jeżeli chcemy znać liczbę wszystkich obiektów danej klasy obecnie istniejących w systemie.

Odpowiedzialność klas

Odpowiedzialność (ang. *responsibility*) jest wyrażona kontraktem lub zobowiązaniem klasy.

Modelując klasy rozpoczyna się zazwyczaj od wyspecyfikowania zobowiązań elementów słownictwa systemu. W trakcie doskonalenia (uściślania) modelu zobowiązania systemu są tłumaczone na realizujący je zbiór operacji i atrybutów.



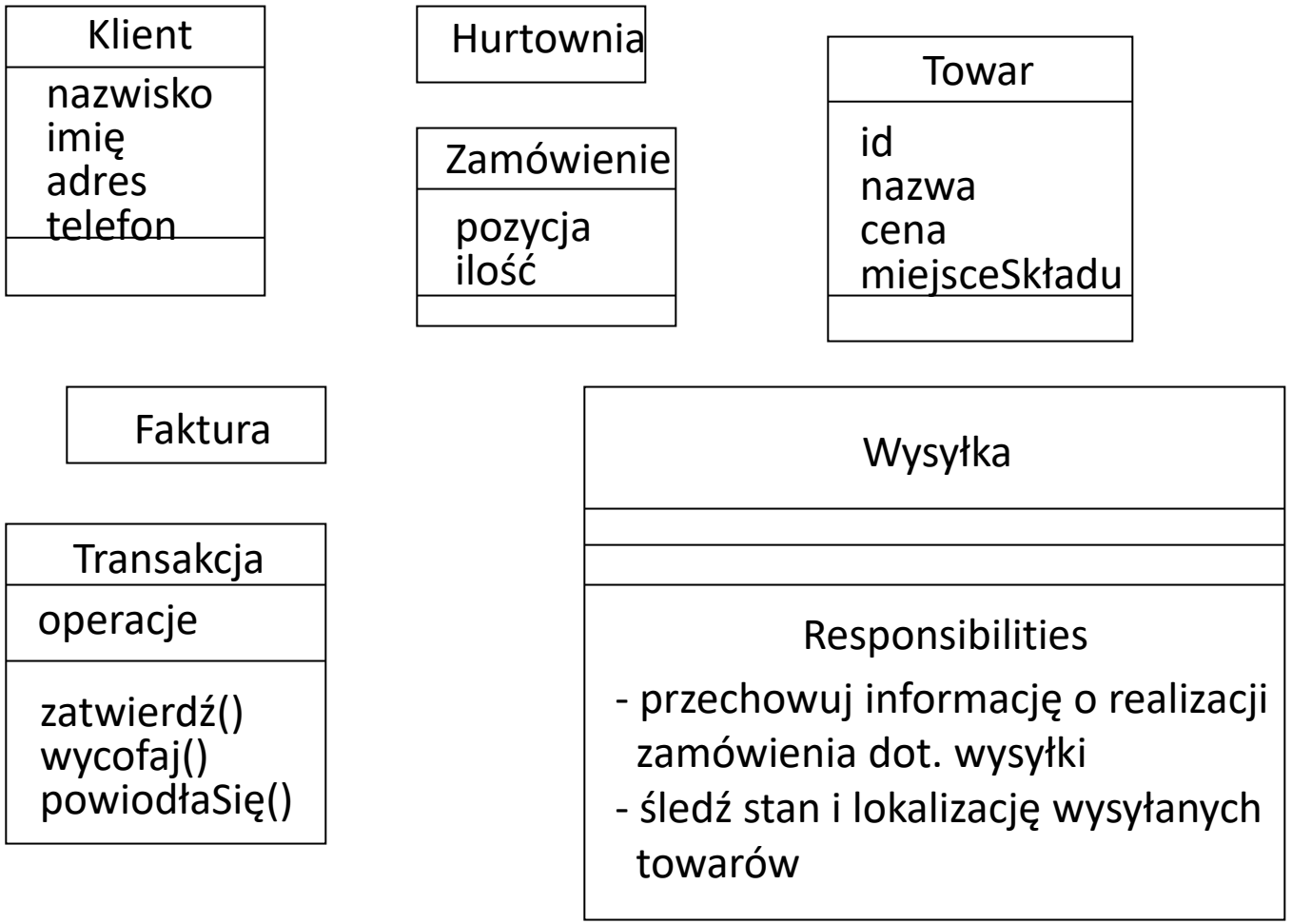
Modelowanie słownictwa systemu

Klasy wykorzystuje się do modelowania abstrakcji pochodzących z dziedziny danego problemu lub technologii rozwiązania. Każda z abstrakcji jest częścią słownictwa systemu – reprezentuje elementy istotne dla użytkowników i twórców systemu.

Wytyczne:

1. Zidentyfikuj elementy, które są stosowane przez użytkowników lub twórców systemu do opisu problemu lub rozwiązania.
2. Ustal zbiór zobowiązań każdej abstrakcji. Sprawdź czy wszystkie klasy są precyzyjnie określone i czy mają równomiernie rozłożone zobowiązania.
3. Uwzględnij atrybuty i operacje potrzebne do wykonania przez daną klasę zobowiązań.

Modelowanie słownictwa systemu – przykład



Modelowanie rozkładu zobowiązań

Uwaga: rozkład zobowiązań powinien być w miarę równomiernie rozłożony między poszczególne klasy.

Wytyczne

1. Zidentyfikuj zbiór klas współpracujących w celu wykonania poszczególnych czynności.
2. Określ zbiór zobowiązań dla każdej klasy.
3. Rozważ zbiór klas jako całość, podziel klasy na mniejsze, jeśli mają zbyt dużo zobowiązań - scal w większe jeśli mają zbyt mało. Przenoś zobowiązania między klasami, aby każda była w pełni samodzielna.
4. Analizuj sposoby wzajemnej kooperacji tych klas i porozdzielaj ich zobowiązania, aby były równomiernie rozłożone.

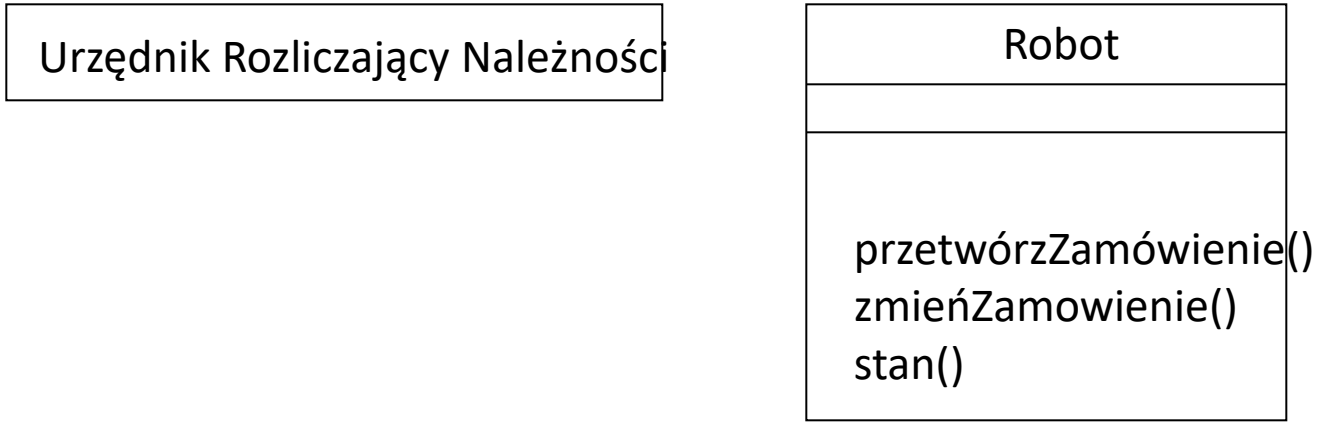
Modelowanie elementów nieprogramowych

Uwaga: może pojawiać się konieczność modelowania elementów nieprogramowych, z poza modelowanego systemu.

Wytyczne

1. Przedstaw każdy element nieprogramowy w postaci klasy.
2. W celu odróżnienia od standardowych bloków UML określ nowy rodzaj przy pomocy stereotypu i określ jego znaczenie i podaj nowy symbol graficzny.
3. Jeśli modelowany element jest sprzętem zawierającym oprogramowanie rozważ możliwość, czy nie można go przedstawić w postaci węzła, aby później rozwijać jego strukturę.

Modelowanie elementów nieprogramowych – przykład



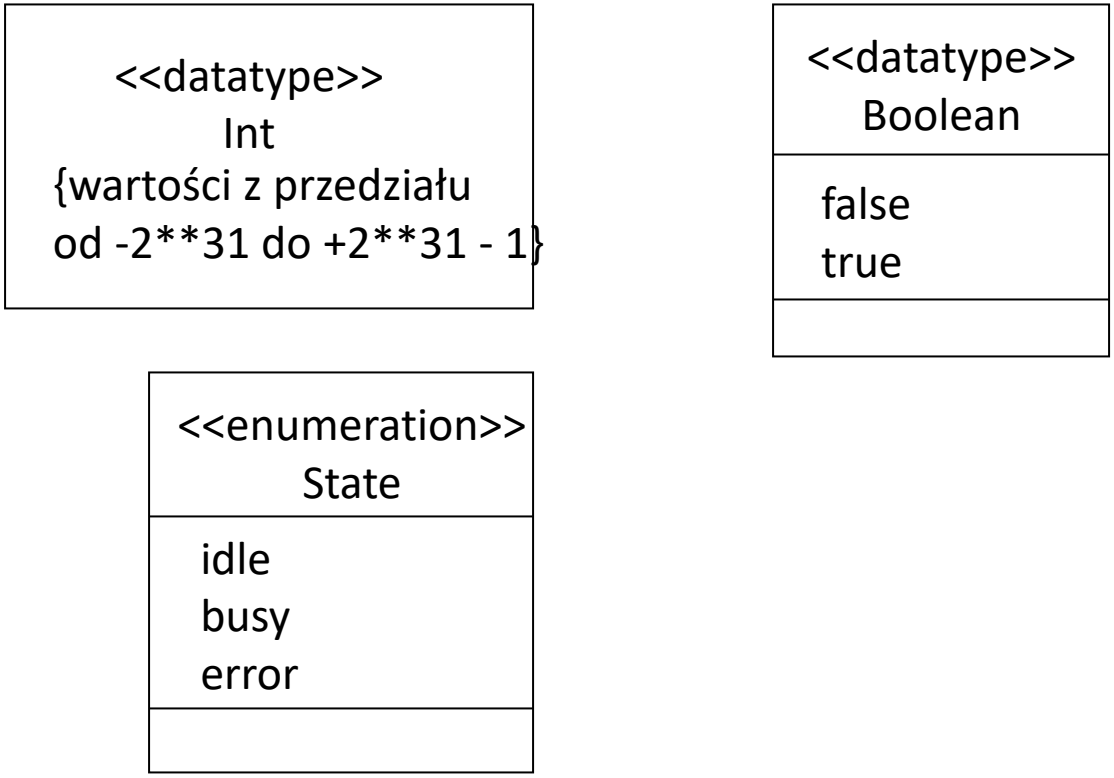
Modelowanie pierwotnych typów danych

Modelowane elementy mogą pochodzić z języka programowania (implementacji). Zwykle są to typy pierwotne (predefiniowane), np. liczby całkowite, znaki, napisy, typy wyliczeniowe itp.

Wytyczne

1. Przedstaw typy jako klasy zaopatrzone odpowiednimi stereotypami.
2. Użyj ograniczeń, jeśli chcesz wyspecyfikować zbiór dopuszczalnych wartości pewnego typu.

Modelowanie pierwotnych typów danych – przykład



Związki między klasami/obiektami

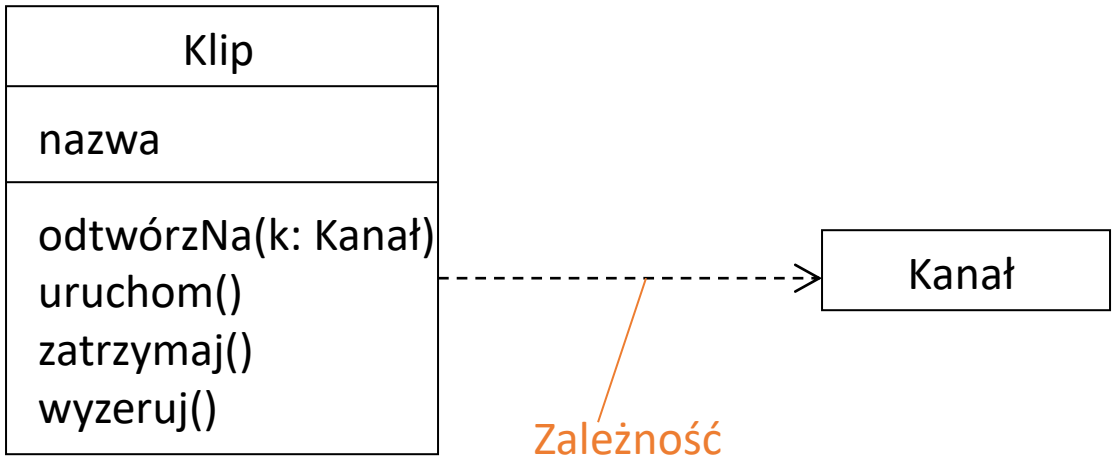
Związek to relacja między elementami. W diagramach UML związki są przedstawiane jako różne (w zależności od rodzaju związku) linie łączące elementy.

Najważniejsze rodzaje związków:

1. **Zależność** (ang. *Dependency*) – często reprezentowana przez relację użycia.
2. **Uogólnienie** (ang. *Generalization*) - związek między klasą ogólną a szczegółową:klasa-podklasa lub potomek-przodek.
3. **Powiązanie** (ang. *Association*) - jest związkiem strukturalnym między elementami klasy.

Zależność między klasami

Zależność – oznacza, że zmiany dokonane w specyfikacji jednego elementu mogą mieć wpływ na inny element, który używa tego pierwszego.



Zależność – uwagi

Zależności są najprostszym i najsłabszym rodzajem relacji łączących klasy.
Przykłady zależności:

- <<call> - operacje w klasie A wywołują operacje w klasie B
- <<create> - klasa A tworzy instacje klasy B
- <<instantiate>> - obiekt jest instancją klasy B
- <<use>> - do zaimplementowania klasy A wymagana jest klasa B

Związek zależności jest wykorzystywany często w sytuacji, gdy używamy klasy, która udostępnia zestaw funkcji narzędziowych ogólnego przeznaczenia.

Uogólnienie między klasami

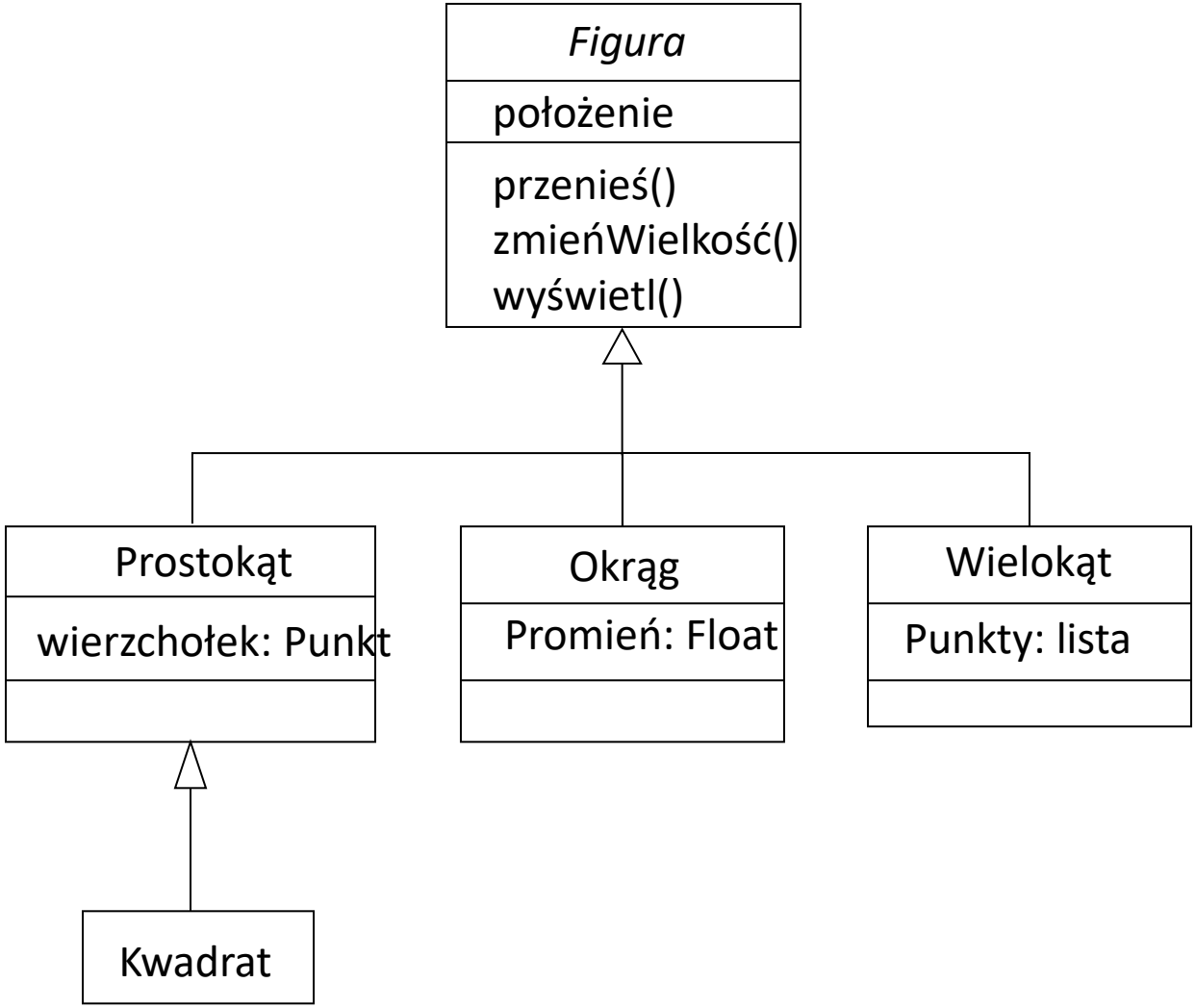
Uogólnienie jest związkiem między elementem ogólnym (nadklasa, przodek) a pewnym specyficznym jego rodzajem (podklasa, potomek).

Uogólnienie polega m.in. na tym, że potomek może wystąpić wszędzie tam gdzie spodziewany jest przodek (lecz nie na odwrót).

Potomek dziedziczy właściwości przodka, w szczególności atrybuty i operacje. Może też mieć własne cechy.

Operacja potomka, mająca tę samą sygnaturę jest ważniejsza (polimorfizm), tzn. „przysłania” operację przodka.

Uogólnienie – przykład



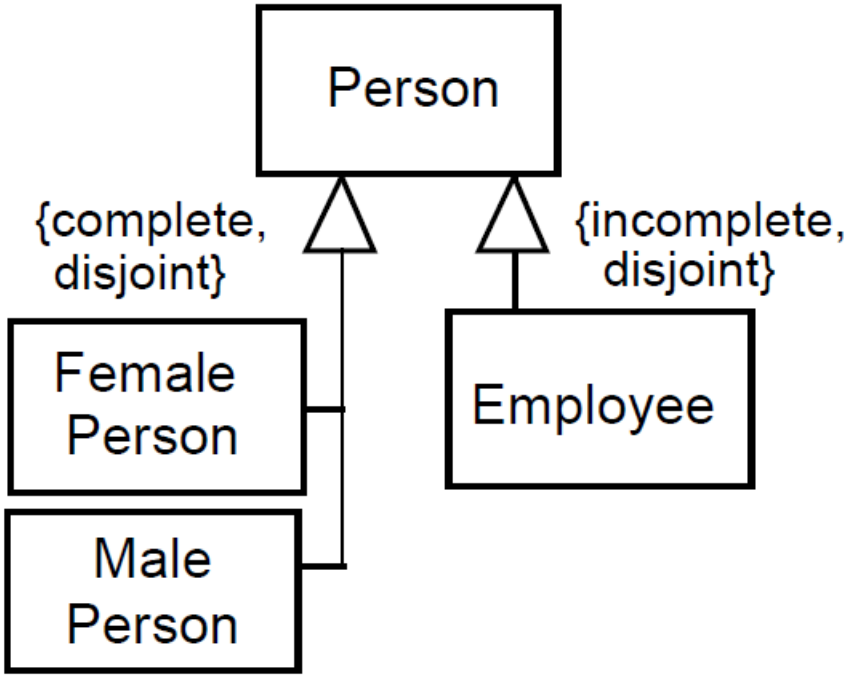
Uogólnienie – zestawy uogólnień

Zestaw uogólnień jest to element, którego instancje definiują kolekcje podzbiorów relacji uogólnień.

Oznaczenia zestawów:

• {complete, disjoint} - Wskazuje, że zestaw uogólnień obejmuje i jego określone klasyfikatory nie mają wspólnej instancje.
• {incomplete, disjoint} - Wskazuje, że zestaw uogólnień nie pokrywa się, a jego określone klasyfikatory nie mają wspólnego instancje.
• {complete, overlapping} - Wskazuje, że zestaw uogólnień obejmuje, a jego określone klasyfikatory mają wspólne cechy instancje.
• {incomplete, overlapping} - Wskazuje, że zestaw uogólnień pokrywa się, a jego określone klasyfikatory mają wspólne instancje.

Zestaw uogólnień – przykład

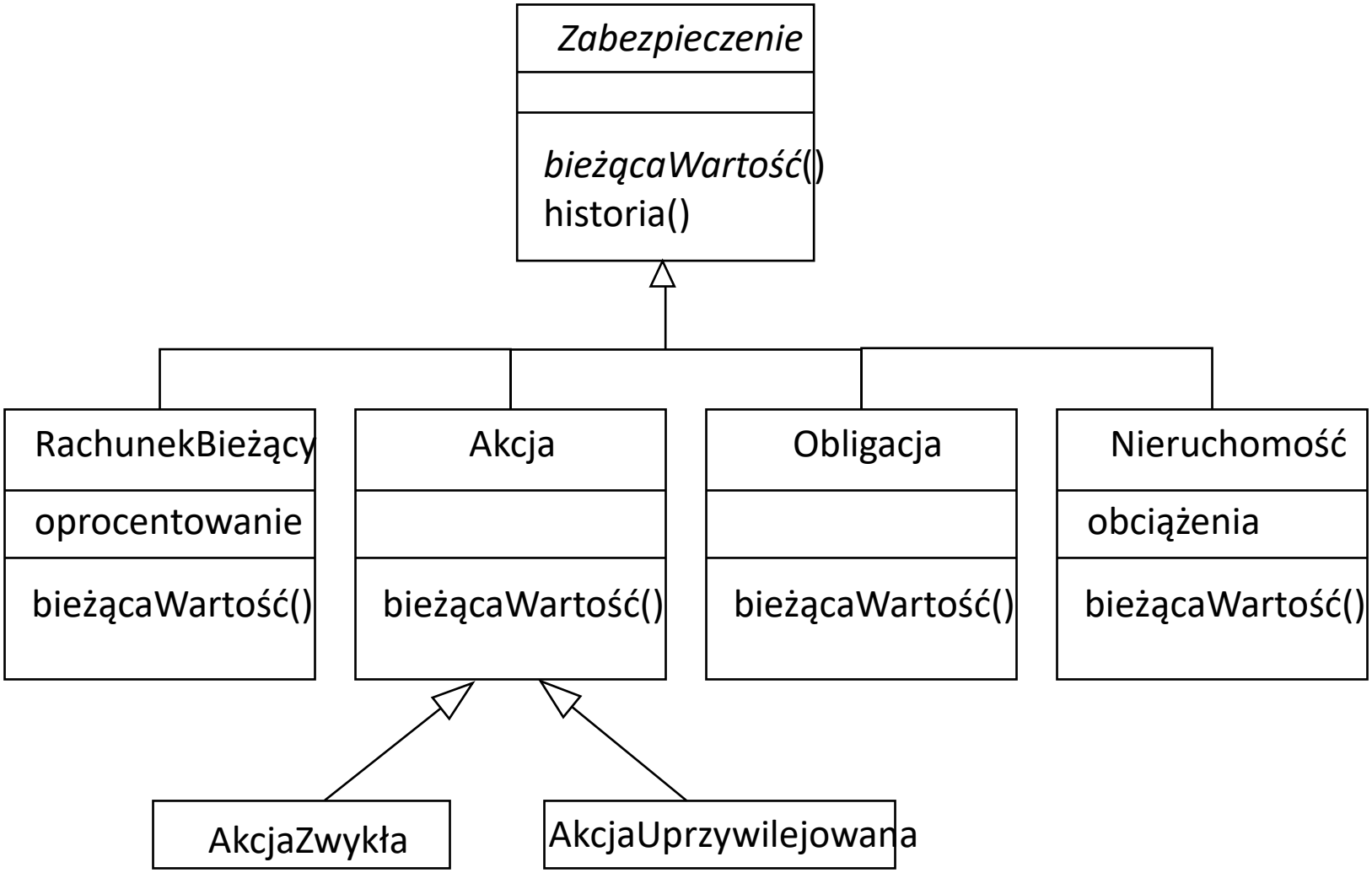


Modelowanie uogólnienia

Wytyczne

1. Poszukaj zobowiązań, atrybutów i operacji wspólnych dla co najmniej dwóch klas.
2. Przenieś wspólne zobowiązania, atrybuty i operacje do klasy ogólnej. Jeśli to konieczne utwórz nową klasę.
3. Zaznacz związki dziedziczenia - od potomka do przodka.

Modelowanie uogólnienia – przykład

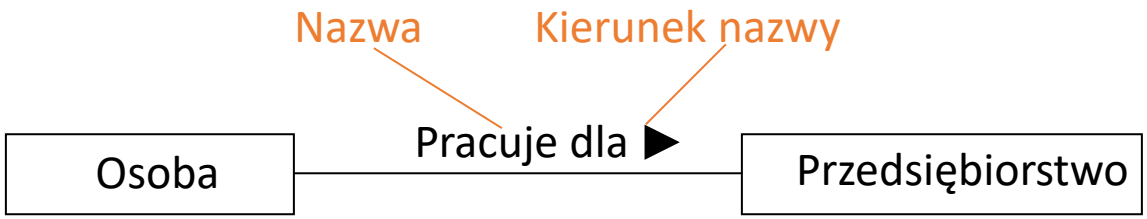


Powiązanie między klasami

Powiązanie to związek strukturalny specyfikujący połączenie obiektów jednego klasyfikatora z obiektami drugiego.

Powiązanie jest związkiem równorzędnych partnerów - klasy są na tym samym poziomie pojęciowym.

Powiązanie między klasami oznacza, że można przejść od obiektu jednej z nich do obiektu drugiej i odwrotnie.



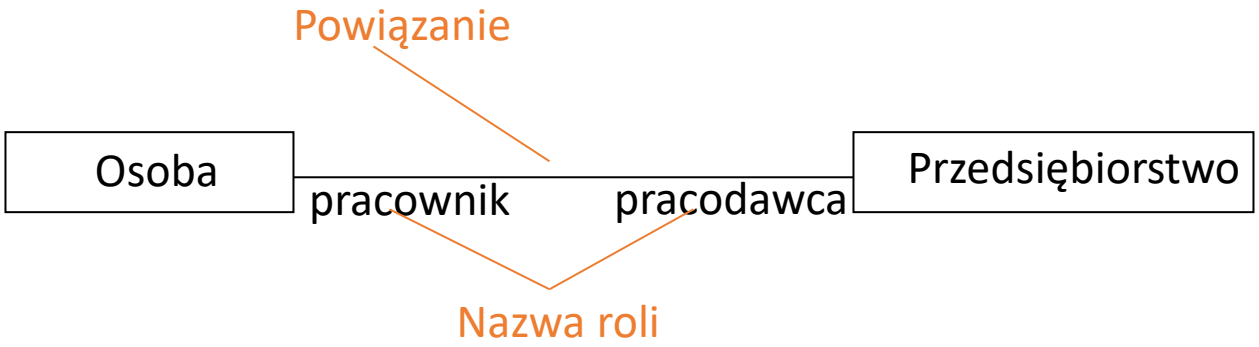
Powiązanie – uwagi i oznaczenia

• Nazwa powiązania może być wyświetlana jako ciąg znaków w pobliżu symbolu powiązania.
• Ukośnik pojawiający się w przed nazwą powiązania lub zamiast nazwy, oznacza, że powiązanie jest pochodny.
• Można umieścić ciąg właściwości w pobliżu symbolu powiązania.

Przykłady:
• {subsets <property-name>} - aby pokazać, że koniec jest podzbiorem właściwości o nazwie <property-name>.
• {redefines <end-name>} - aby pokazać, że koniec przedefiniowuje nazwę o nazwie <end-name>.
• {union} - aby pokazać, że koniec jest pochodny ze związków z jego podzbiorów.
• {ordered} - aby pokazać, że koniec reprezentuje uporządkowany zbiór.
• {nonunique} - aby pokazać, że koniec reprezentuje kolekcję, która pozwala, aby ten sam element pojawił się więcej niż raz.
• {sequence} or {seq} - pokazują, że koniec reprezentuje sekwencję.

Role

Rola zachowanie bytu w ustalonym kontekście.



Innymi słowy rola, to pewne „oblicze”, które obiekty klasy przy jednym końcu prezentują obiektom klasy przy drugim końcu.

Na rysunku *Osoba* pełniąca rolę *pracownika* jest powiązana z *Przedsiębiorstwem* będącym w roli *pracodawcy.*

Powiązanie a krotności

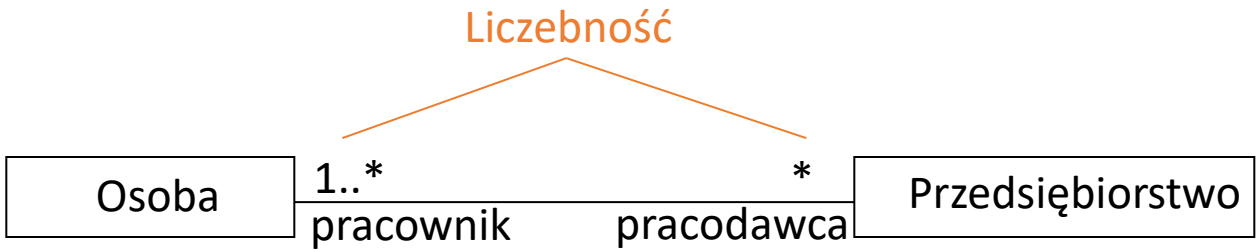
Krotność pozwala określić minimalną i maksymalną liczbę obiektów, jakie można powiązać z daną cechą.

Ta wielokrotność (*ile*) nazywana jest **liczebnością** (ang. *multiplicity*).

Dolna granica … górna granica – przedział od-do

- 1 – dokładnie jeden obiekt
- 0…1 – opcjonalnie jeden obiekt
- 1…* - przynajmniej jeden obiekt
- 1,3,5 – konkretne liczby obiektów
- * - dowolna liczba obiektów

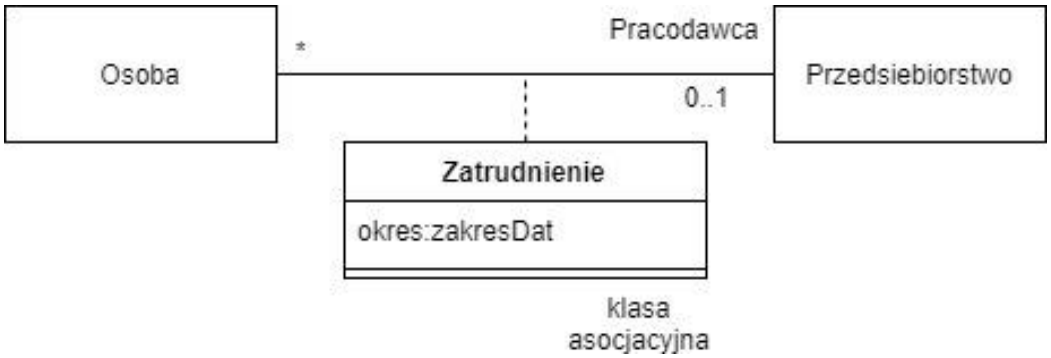
Krotności/liczebności – przykład



Przy modelowaniu często zachodzi potrzeba określenia **ile** obiektów może być połączonych przez jeden egzemplarz powiązania

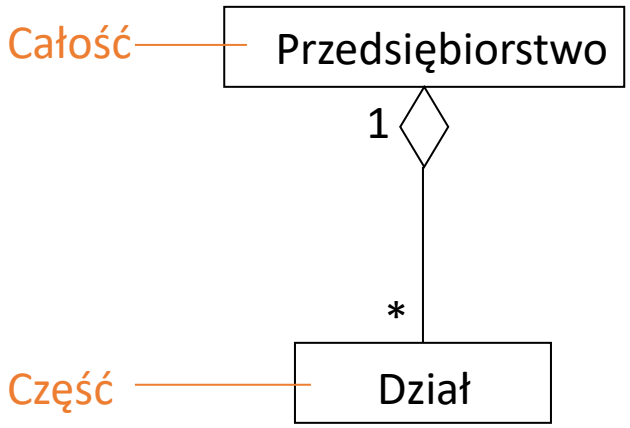
Klasa asocjacyjna

Klasy asocjacyjne są związane z relacją asocjacji i opisują jej właściwości. Mają dostęp do innych obiektów uczestniczących w relacji. Informacje przechowywane w klasie asocjacyjnej nie są związane z żadną z klas uczestniczących w asocjacji, dlatego wygodnie jest stworzyć dodatkową klasę i powiązać ją z relacją. Klasy asocjacyjne są reprezentowane graficznie jako klasy połączone linią przerywaną z relacją asocjacji, której dotyczą.



Agregacja jako powiązanie

Agregacja oznacza związek *całość-część*, tzn. dana klasa (całość) składa się z mniejszych (części).



Agregacja częściowa

- Wykorzystywana jest do zaznaczenia, że jedna klasa w rzeczywistości posiada obiekty innej, ale może je jednocześnie współdzielić.
- Agregacja jest silniejszą formą asocjacji. W przypadku tej relacji równowaga między powiązanymi klasami jest zaburzona: istnieje właściciel i obiekt podrzędny, które są ze sobą powiązane czasem swojego życia.
- Właściciel jednak nie jest wyłącznym właścicielem obiektu podrzędnego, zwykle też nie tworzy i nie usuwa go.
- Przedstawiana jest przy użyciu strzałki zakończonej pustym rombem tuż obok klasy zawierającej.

Agregacja całkowita

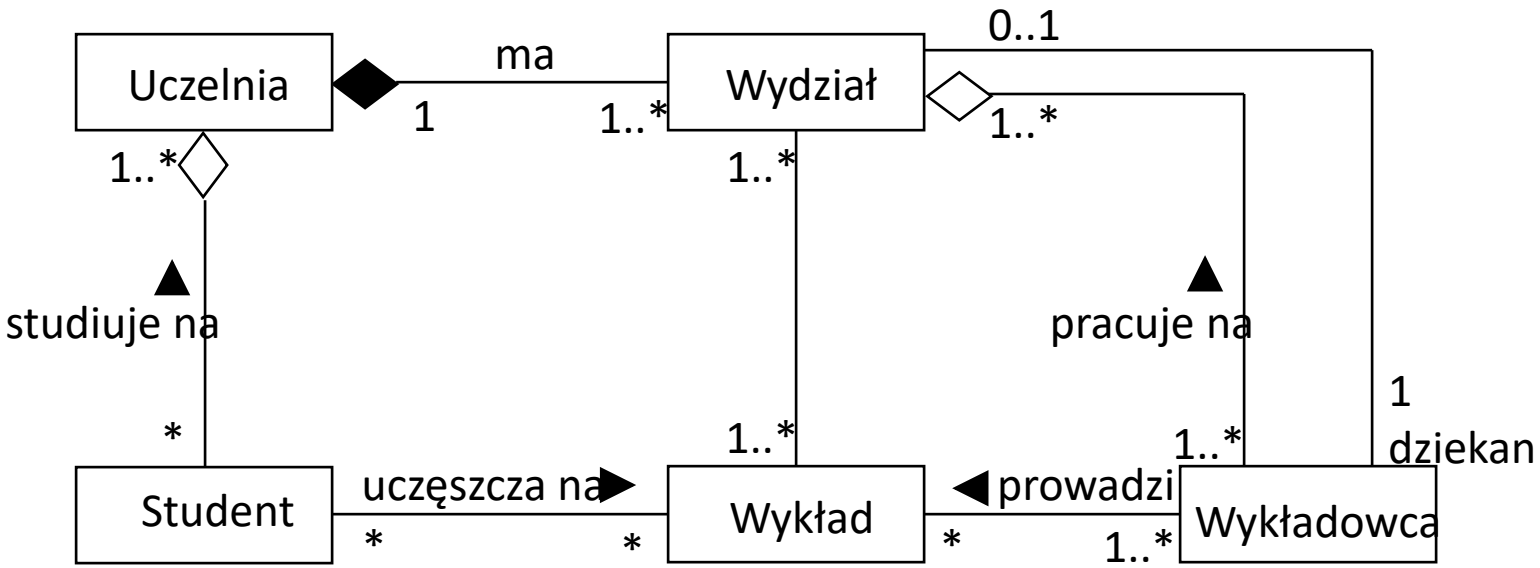
- Reprezentowana jest przez strzałkę zakończoną wypełnionym rombem.
- Polega ona na modelowaniu wewnętrznych części klasy.
- Agregacja całkowita jest najsilniejszą relacją łączącą klasy. Reprezentuje relacje całość-część, w których części są tworzone i zarządzane przez obiekt reprezentujący całość.
- Ani całość, ani części nie mogą istnieć bez siebie, dlatego czasy ich istnienia są bardzo ściśle ze sobą związane i pokrywają się: w momencie usunięcie obiektu całości obiekty części są również usuwane.

Modelowanie związków strukturalnych

Wytyczne

1. Dla każdej pary klas poszukaj, możliwości przechodzenia obiektów jednej z nich do obiektów drugiej. Dla takich klas określ powiązanie. Identyfikacja powiązania w oparciu o dane.
2. Analizuj, czy dla każdej pary klas konieczna jest interakcja między obiektami jednej a obiektami drugiej, inna niż przekazywanie ich jako parametrów. Dla takich klas określ powiązanie. Identyfikacja powiązania w oparciu o zachowanie.
3. Dla każdego powiązania określ liczebność i nazwy ról (ułatwiają zrozumienie modelu).
4. Jeśli jakaś klasa stanowi strukturalną lub organizacyjną całość w stosunku do klas z drugiego końca związku, to zaznacz powiązanie jako agregację.

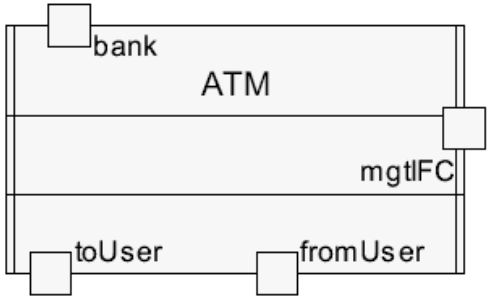
Modelowanie związków strukturalnych – przykład



Porty w klasach

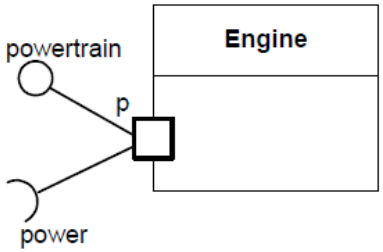
Porty to miejsca przez które jest realizowana interakcja klasy aktywnej, to inaczej nazwane miejsca interakcji. Porty mogą być dziedziczone.

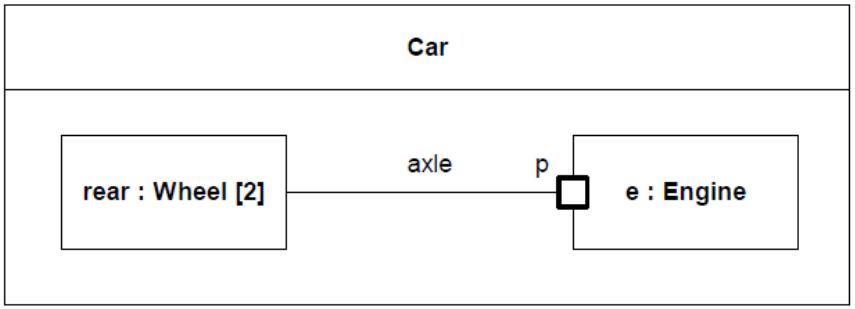
Port oznaczony jest małym kwadratem obok którego znajduje się nazwa portu.



Możemy także wyróżnić porty **behawioralne** i **niebehawioralne**. Pierwsze z nich związane są bezpośrednio z maszynami stanowymi – wszystkie sygnały przechodzące przez port są konsumowane przez zachowanie klasy. Drugi rodzaj wymaga ustanowienia połączenia.

Porty – przykłady



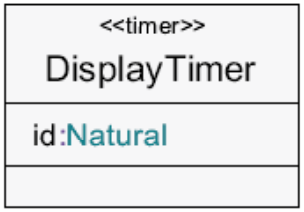


Stereotypy – wprowadzenie

- Są podstawowym mechanizmem rozszerzania języka UML.
- Jeżeli okaże się, że jest potrzebny mechanizm modelowania, który nie istnieje w UML-u, ale jest podobny do czegoś, co już istnieje, to można potraktować nowy mechanizm jako stereotyp mechanizmu UML.
- Przykładem stereotypu jest interfejs.
- Stereotypy są zwykle oznaczane tekstem w podwójnych nawiasach kątowych, przykład <<interface>>
- Ale można je również reprezentować, definiując dla nich specjalną ikonę.

Stereotypy – przykłady

Sygnały – to asynchroniczne wiadomości przesyłane pomiędzy klasami aktywnymi. Można także definiować listy sygnałów.



Zdarzenia typu **timer** – zdarzenia w istocie podobne do sygnałów.

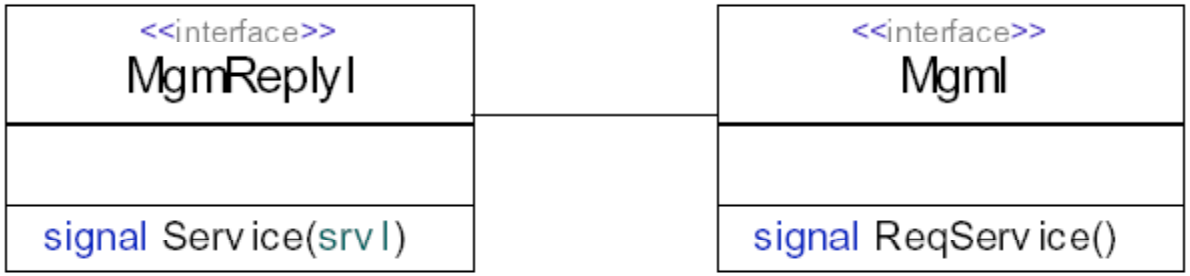
A także inne.

Interfejsy w klasach

Razem z portami mogą występować także interfejsy.

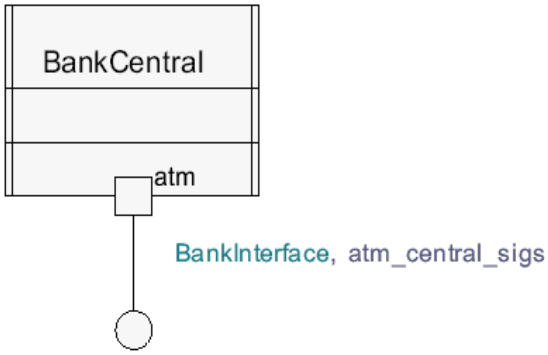
Interfejs – grupa atrybutów i operacji, które muszą być zaimplementowane w danej klasie.

Operacje interfejsu zazwyczaj opisują pewne usługi oferowane przez daną klasę. Poza operacjami interfejs może zawierać atrybuty, a także sygnały.

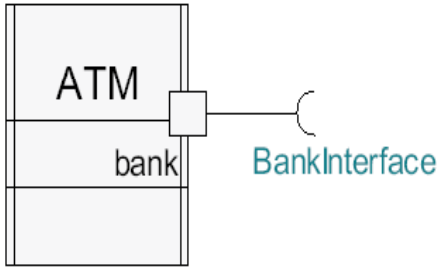


Dwie kategorie interfejsów

Istnieją dwa rodzaje interfejsów.



Interfejs **realizowany** (lub **implementowany**) – interfejs który klasa realizuje przez port.



Interfejs **wymagany** (lub **realizowany**) – pokazuje jakie sygnały wychodzące z powinny być obsłużone.

Klasy abstrakcyjne

Klasa abstrakcyjna deklaruje wspólną funkcjonalność grupy klas. Nie może posiadać obiektów i musi definiować podklasy. Nazwy klas pisze się wtedy kursywą lub można zastosować ograniczenie {abstract}

Klasy abstrakcyjne a interfejsy

• Celem tworzenia klas abstrakcyjnych i interfejsów jest identyfikacja wspólnych zachowań różnych klas, które są realizowane w różny od siebie sposób. Użycie tych mechanizmów pozwala na wykorzystanie relacji uogólniania do ukrywania (hermetyzacji) szczegółów implementacji poszczególnych klas.
• Klasa abstrakcyjna reprezentuje wirtualny byt grupujący wspólną funkcjonalność kilku klas. Posiada ona sygnatury operacji (czyli deklaracje, że klasy tego typu będą akceptować takie komunikaty), ale nie definiuje ich implementacji.
• Podobną rolę pełni interfejs. Różnica polega na tym, że klasa abstrakcyjna może posiadać implementacje niektórych operacji, natomiast interfejs jest czysto abstrakcyjny (choć, oczywiście interfejs i klasa w pełni abstrakcyjna są pojęciowo niemal identyczne).
• Ponieważ klasy abstrakcyjne nie mogą bezpośrednio tworzyć swoich instancji (podobnie jak interfejsy, które z definicji nie reprezentują klas, a jedynie ich typy) dlatego konieczne jest tworzenie ich podklas, które zaimplementują odziedziczone abstrakcyjne metody. W przypadku interfejsu sytuacja jest identyczna.

Szablony klas

Szablon klasy określa, z jakimi innymi klasami (nie obiektami!) może współpracować podana klasa. Klasy te są przekazywane jako jej parametry. Klasa będąca uszczegółowieniem takiej klasy jest **klasą związaną.** Szablony klas to pojęcie wywodzące się z języka C++. Oznaczają one klasy, których definicja wymaga podania argumentów będących innymi klasami. W ten sposób szablon klasy jest swego rodzaju niepełną klasą, która dopiero po ukonkretnieniu może zostać użyta.

```
class Zbiór <T>{
void wstaw(T nowyElement);
void usuń(T element)
}
```

Klasy i diagramy – uwagi końcowe

- Model, który jest tworzony, powinien być dostosowany do etapu/fazy projektu.
- Na etapie analizy należy rysować modele pojęciowe.
- Pracując nad oprogramowaniem, należy skoncentrować się na modelach specyfikacyjnych.
- Modele implementacyjne należy rysować tylko wtedy, gdy trzeba zilustrować jakąś technikę implementacyjną.
- Nie należy rysować modeli dla wszystkiego, zamiast tego warto się skoncentrować na najważniejszych obszarach.