# Język UML

dr inż. Piotr Szwed
C3, pok. 212
e-mail: pszwed@ia.agh.edu.pl
http://pszwed.ia.agh.edu.pl

# Diagramy klas

# Diagramy klas

Diagramy klas są najbardziej charakterystycznym elementem wszystkich języków graficznych modelowania obiektowego. Przedstawiają one statyczne elementy należące do pewnej dziedziny (klasy) oraz związki pomiędzy nimi.

| **Obiekt** |
| --- |
| Obiektem jest każdy byt – pojęcie lub rzecz – mający znaczenie w kontekście rozwiązywania problemu w danej dziedzinie przedmiotowej. |

# Cechy obiektów

Każdy obiekt posiada trzy podstawowe cechy:

- **Tożsamość** - unikalny uchwyt, dzięki któremu obiekt odróżniany jest od innych.
- **Stan** - zawartość komórek przechowujących dane opisujące obiekt.
- **Sposób zachowania** - zbiór akcji, które potrafi wykonywać obiekt.

# Dziedzina przedmiotowa

W zależności od modelu obiekt może reprezentować:

- element dziedziny problemu
- element programu

# Element dziedziny problemu

Systemy, które modelujemy przetwarzają dane. Dane te odpowiadają pojęciom pojawiającym się w opisie dziedziny problemu. Są to na przykład:

- dane klienta, towar, zamówienie, faktura dla sklepu
- dane pojazdu (rok produkcji, marka, typ, numer nadwozia, data rejestracji) dane właściciela (imię, nazwisko, PESEL, adres) dla systemu rejestracji pojazdów
- dane pojazdu (marka, model, elementy wyposażenia, cena) dane klienta (imię, nazwisko, adres) dla systemu obsługi komisu samochodowego
- zdarzenie wygenerowane przez czujnik, zmierzona wartość, czas pomiaru dla systemu sterowania

# Element programu

Obiekt może reprezentować element składowy programu. Jest on odpowiedzialny za przechowywanie, przetwarzanie i przesyłanie danych. Przykładami mogą być:

- dokument, widok
- okno dialogowe
- baza danych (lub jej obiektowe opakowanie)
- wątek, kolejka komunikatów, semafor

# Klasa 1

Dowolny obiekt jest instancją abstrakcyjnego pojęcia, jakim jest *klasa*. Podstawę identyfikacji klas stanowią grupy obiektów charakteryzujących się:

- identyczną strukturą danych, czyli takimi samymi atrybutami
- identycznym zachowaniem, czyli takimi samymi operacjami
- identycznymi związkami
- identycznym znaczeniem w określonym kontekście

# Klasa 2

| **Klasa** |
| --- |
| Jest uogólnieniem zbioru obiektów, które mają takie same atrybuty, operacje, związki i znaczenie. |

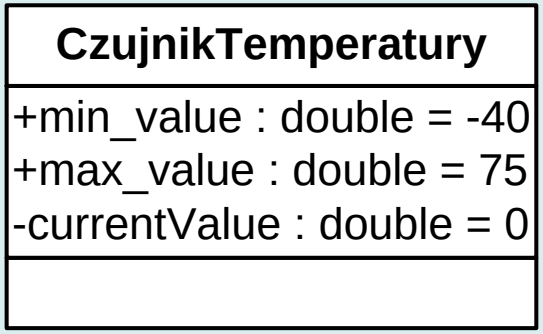
Klasa także zawiera opis, który umożliwia stworzenie obiektów należących do danej klasy.

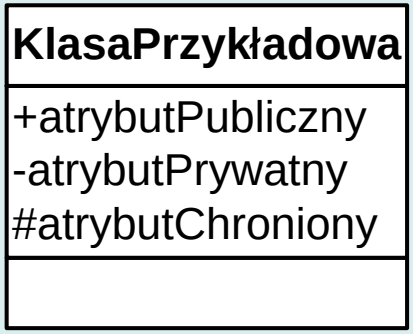
# Atrybut

**Atrybut – jest nazwaną własnością (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.

# Przykłady klas z atrybutami

# Operacja

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

- Klasa może mieć dowolną liczbę operacji lub nie mieć ich wcale.
- Diagram klas nie specyfikuje, w jaki sposób jest zaimplementowana usługa, ale pokazuje sygnatury operacji zdefiniowanych dla danej klasy, czyli ich interfejs.

| CzujnikProgowy |
| --- |
| |
| +ustawPoziom(in prog : double)<br>+ustawOdbiorceSygnalu(in odbiorca)<br>+odczytajStan() : bool |

| Dialog |
| --- |
| |
| +onOK()<br>+onCancel() |

| Stos |
| --- |
| |
| +push(in obj : object)<br>+pop() : object |

# Odpowiedzialność

- **Odpowiedzialność** (responsibility) jest wyrażona kontraktem lub zobowiązaniem typu lub klasy.
- Odpowiedzialność jest formą specyfikacji zadań, jakie klasa będzie pełniła w rozwijanym modelu. Zazwyczaj jest ona jawnie lub niejawnie określana przed przydzieleniem klasie atrybutów i operacji (czyli przed podjęciem pewnych decyzji projektowych, które określają, jak dana odpowiedzialność zostanie zrealizowana).

# Odpowiedzialność - przykłady

Klasa Klient odpowiada za przechowywanie danych klienta: imienia, nazwiska, adresu i numeru telefonu.

| Klient |
| --- |
| +imie : string<br>+nazwisko : string<br>+adres : string<br>+telefon : string |
| |

Klasa Pojazd odpowiada za symulację zachowania pojazdu w modelowanym systemie, stąd odpowiednie atrybuty i operacje:

| Pojazd |
| --- |
| -x : double<br>-y : double<br>-dx : double<br>-dy : double<br>-predkosc : double<br>-przyspieszenie : double |
| +rysuj(in graphics) |

# Związki 1

W paradygmacie programowania obiektowego program może być traktowany jako sieć obiektów, które wysyłają do siebie komunikaty. Typową implementacją komunikatu jest wywołanie metody.

Aby wysłać właściwy komunikat do właściwego odbiorcy obiekt musi znać odbiorcę, znać jego interfejs. Z tego powodu konstruując program definiujemy związki pomiędzy klasami.

# Związki 2

Związki także mogą wynikać ze zobowiązań klas ustalanych w trakcie analizy dziedziny. Na przykład klasa Faktura musi przechowywać informacje o Nabywcy, Sprzedawcy i zakupionych Towarach lub Usługach.

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

# Rodzaje związków

W języku UML wyróżnia się trzy rodzaje związków:

- **Zależność** (ang. *dependency*) – definicja klasy lub jej implementacja wymaga informacji o innej klasie
- **Uogólnienie** (ang. *generalization*) -związek między klasą bardziej ogólną a klasą wyspecjalizowaną (w językach obiektowych implementowany jako dziedziczenie)
- **Powiązanie** (ang. *association*) - związek strukturalny pomiędzy klasami odpowiadający sytuacji, gdy obiekt danej klasy przechowuje informacje o obiektach innej klasy.

# Zależność 1

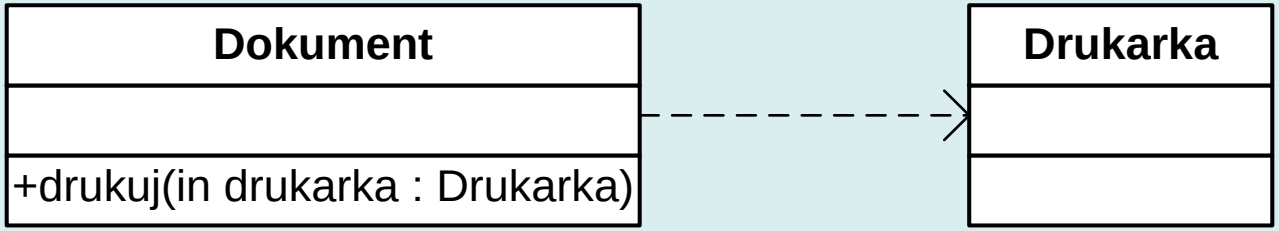
- Zależność jest związkiem użycia. Zmiany dokonane w specyfikacji elementu mają wpływ na inny element, który go używa (ale niekoniecznie na odwrót). Na diagramie związek ten jest przedstawiany jako linia przerywana z grotem skierowanym na element, od którego coś zależy.
- Użycie nie oznacza konieczności przechowywania informacji o obiekcie innej klasy. Kiedy klasa dziedziczy po innej klasie lub przechowuje informacje o obiekcie innej klasy, mimo że *używa* innej klasy stosujemy mocniejsze powiązanie (uogólnienie lub asocjację).

# Zależność 2

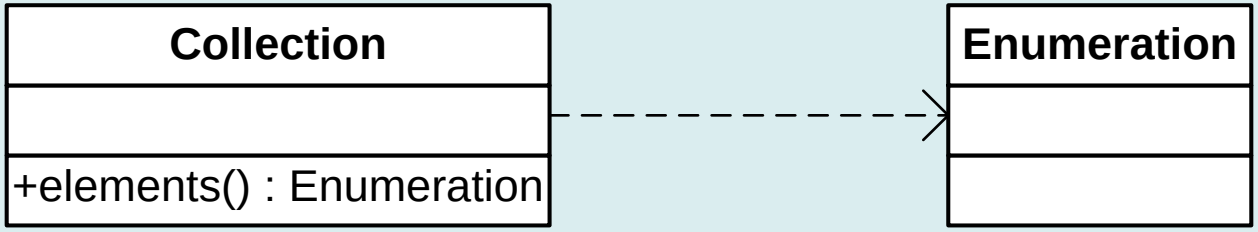
- Zazwyczaj związek zależności rezerwowany jest dla sytuacji, kiedy formalnym parametrem metody lub zwracaną wartością jest obiekt (grupa obiektów) innej klasy.
- W językach programowania na poziomie kodu źródłowego zależność może być wyrażana jako dyrektywa *#include* preprocesora (C++) lub *import* (Java).

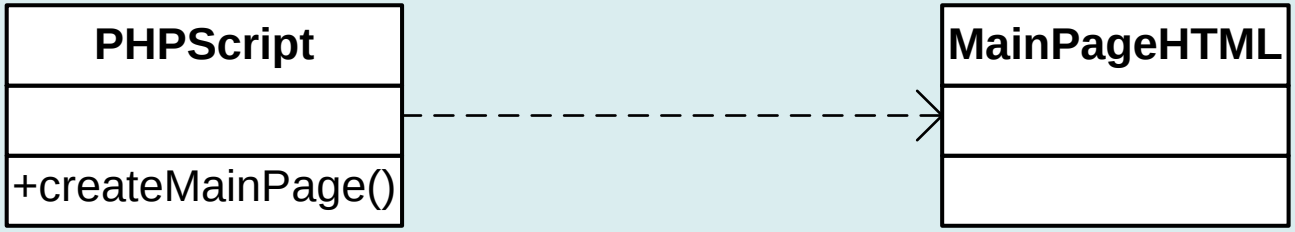
# Zależność 3

- Zależność związana z formalnym parametrem metody




- Zależność związana z typem zwracanej wartości




Ostatni przykład przedstawia zależność pomiędzy skryptem generującym stronę HTML a obiektem reprezentującym stronę (skrypt musi wiedzieć, jak wygenerować stronę).

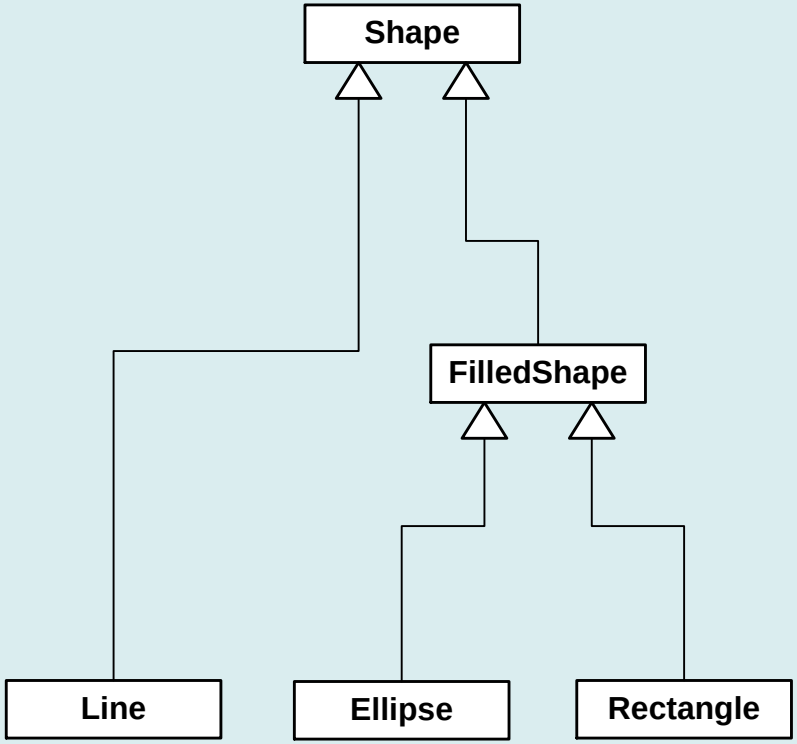
# Uogólnienie 1

- Uogólnienie jest związkiem między *elementem ogólnym* (nazywanym nadklasą, przodkiem, klasą bazową) a pewnym *specyficznym* jego *rodzajem* (nazywaną podklasą, klasą potomną, klasą pochodną).
- Uogólnieniu odpowiada fraza języka naturalnego *jest*, *jest rodzajem*.

Przykłady:

- SamochódOsobowy jest Pojazdem
- Słoń jest Zwierzęciem
- Stół jest Meblem jest Przedmiotem

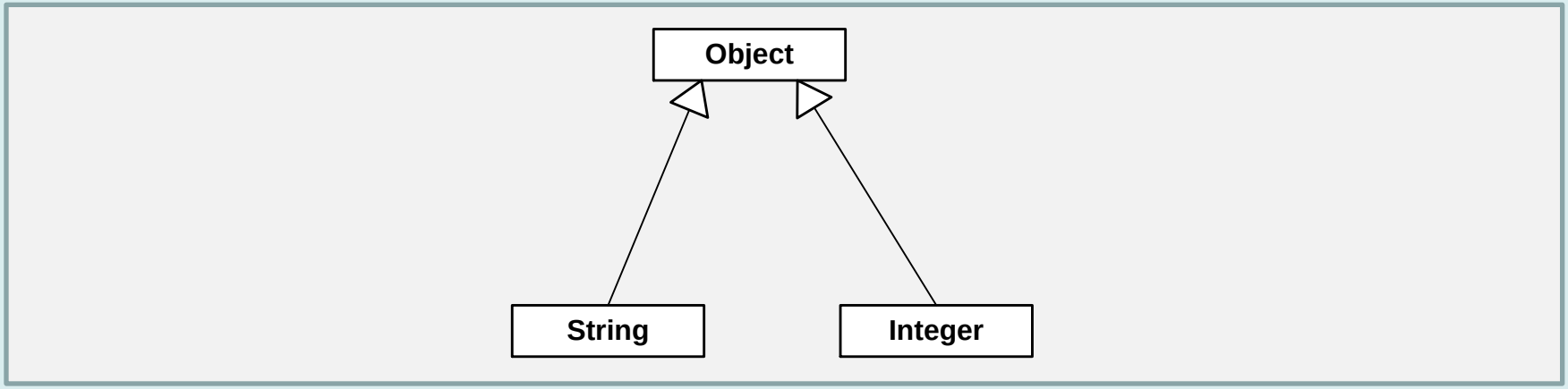
# Uogólnienie 2

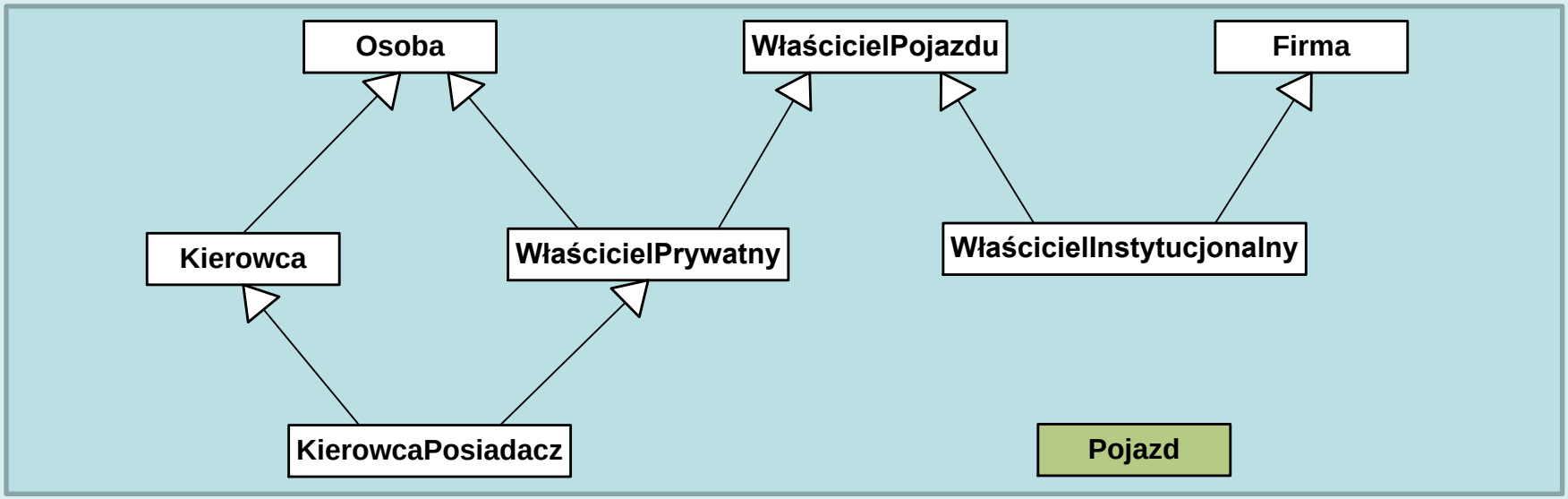



- Grupa klas powiązanych relacją uogólnienia tworzy hierarchię.
- Klasę najbardziej ogólną zazwyczaj umieszczamy u góry.

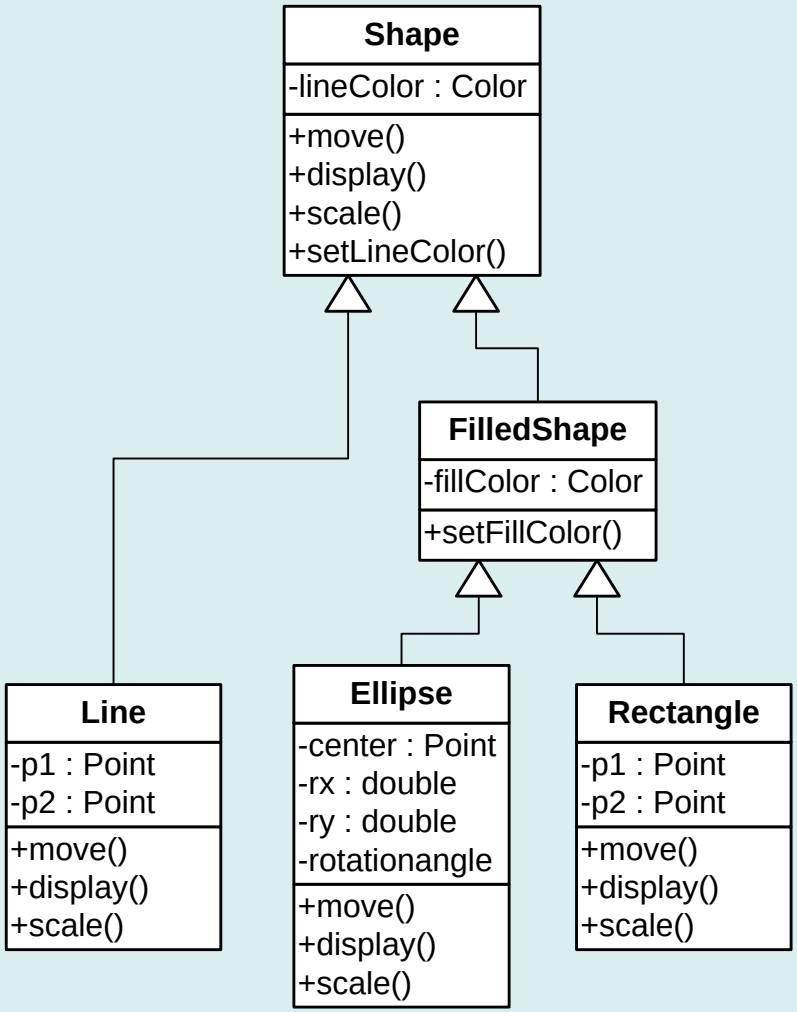
# Uogólnienie 3

Klasa może mieć jednego przodka, kilku przodków lub nie mieć ich wcale.



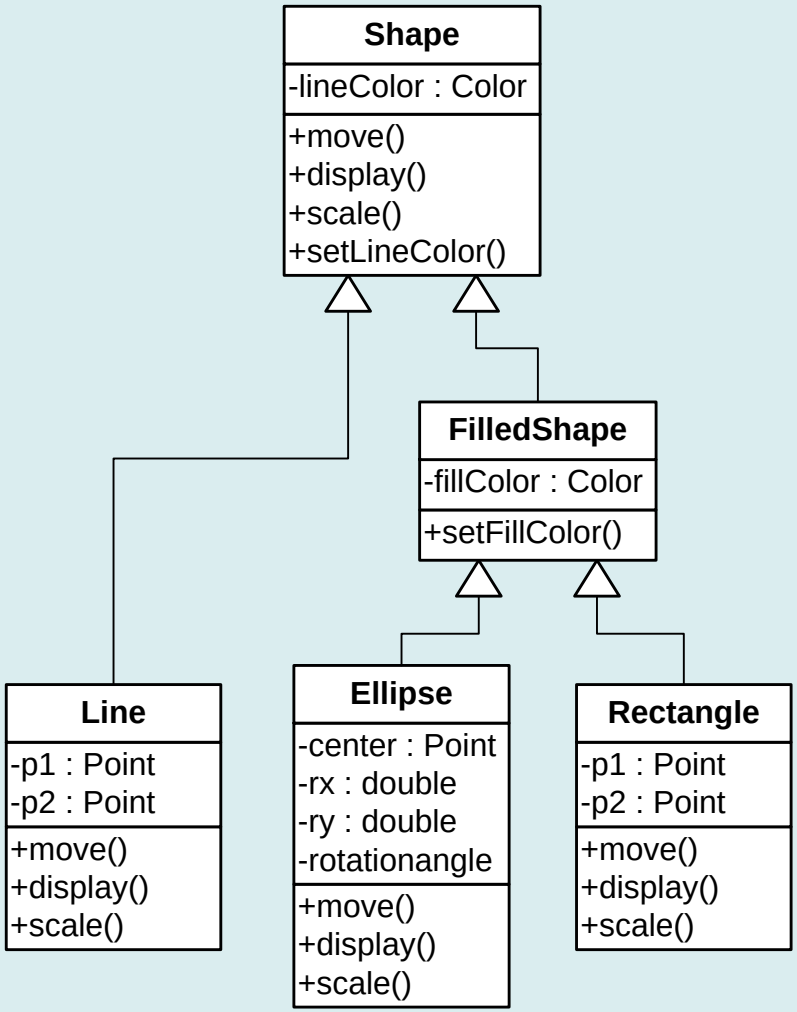



# Uogólnienie 4




- Potomek dziedziczy właściwości przodka, w szczególności atrybuty i operacje.
- Równocześnie może dodawać atrybuty i przedefiniowywać operacje.

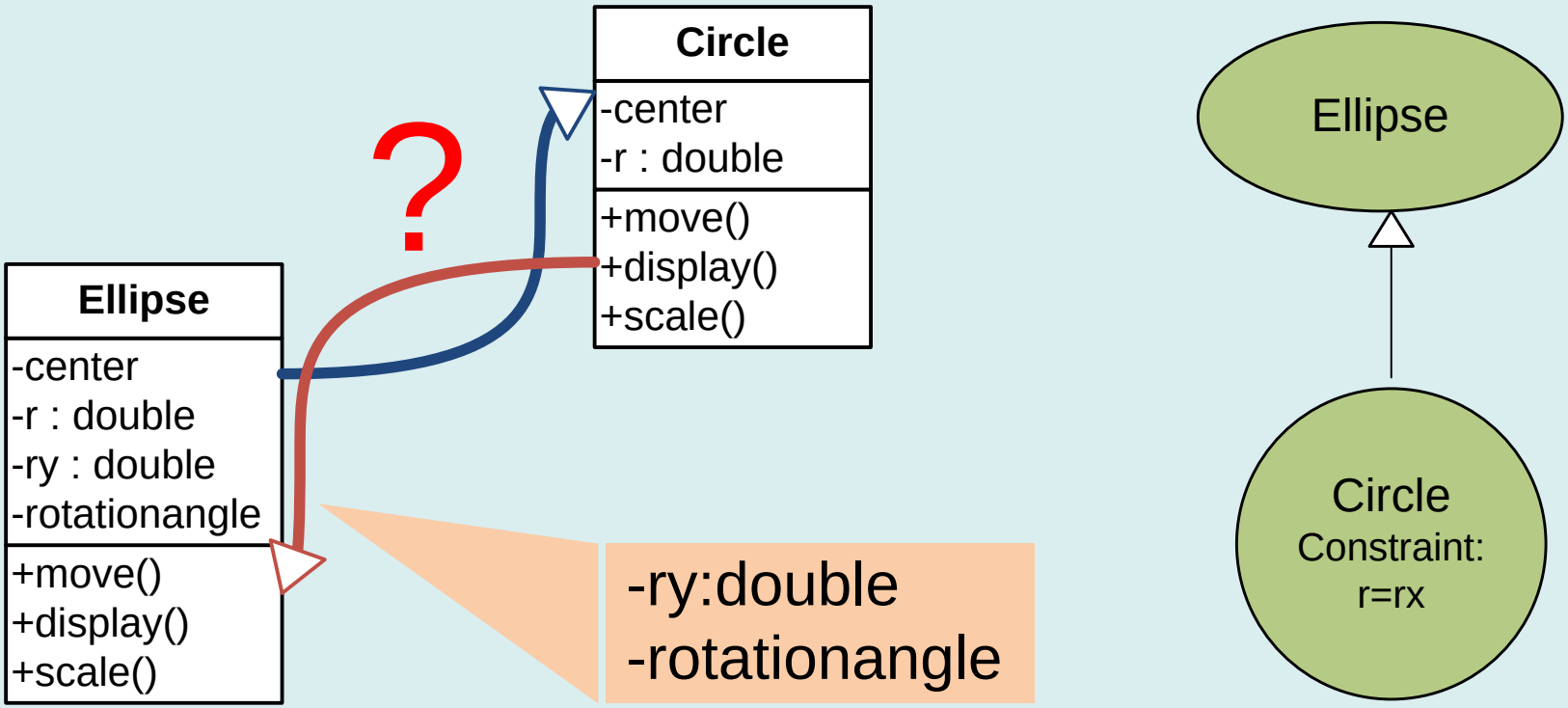
# Uogólnienie 5




- Operacja potomka, mająca tę samą sygnaturę jest ważniejsza niż operacja przodka (*polimorfizm*).
- Potomek może wystąpić wszędzie tam gdzie spodziewany jest przodek, lecz nie na odwrót. (automatyczne rzutowanie w górę, *upcasting*)

# Uogólnienie 6

Pułapki dziedziczenia?



# Asocjacja 1

Asocjacja (powiązanie) to związek strukturalny pomiędzy klasami, który wskazuje, że ich obiekty są ze sobą połączone.

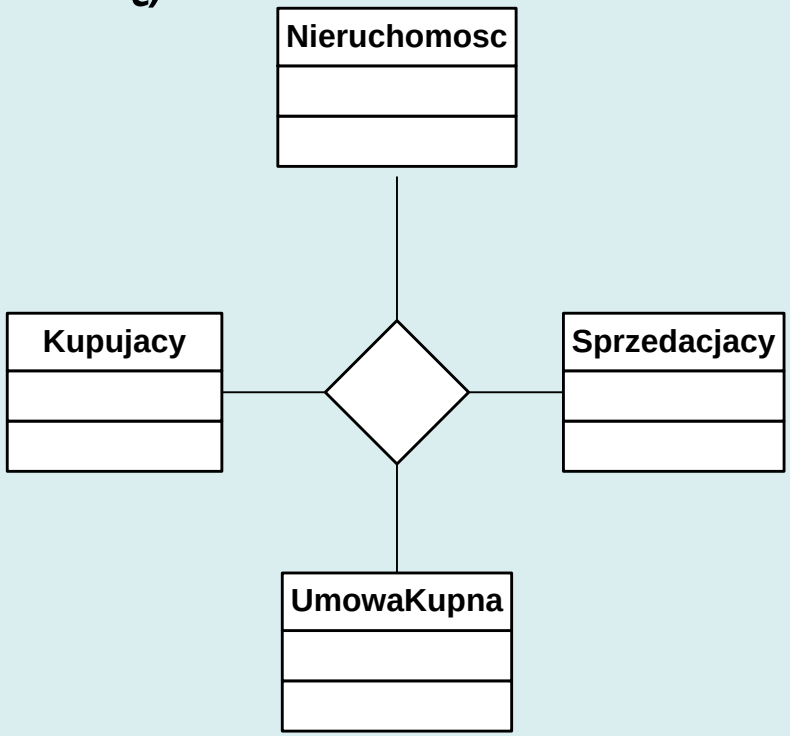
- Istnienie powiązania między klasami oznacza, że możliwe jest przejście (nawigacja) pomiędzy obiektami.
- Aby to było możliwe, obiekty muszą przechowywać informacje o innych powiązanych obiektach – w postaci przeznaczonych specjalnie do tego atrybutów.

# Asocjacja 2

Asocjacja może być relacją

- dwuargumentową (binarną)
- wieloargumentową (n-arną).


Asocjacja binarna

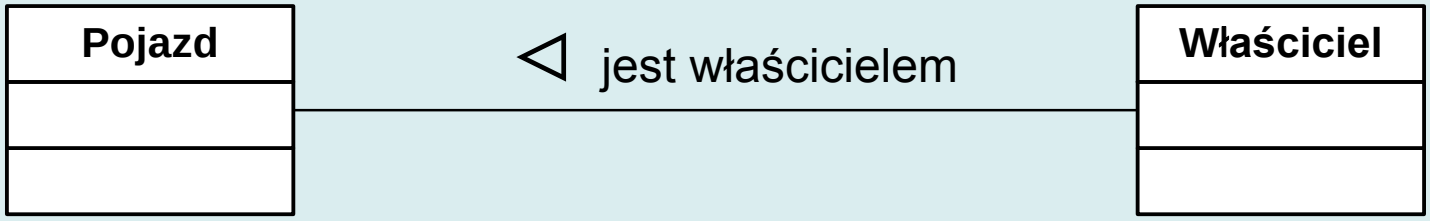



Asocjacja n-arna

# Asocjacja 3: Nazwa

**Nazwa**

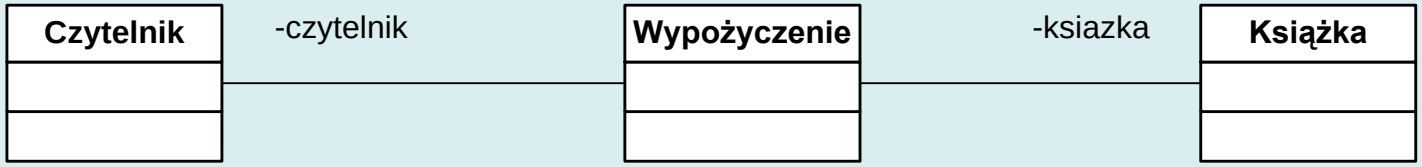
Asocjacja może mieć nazwę określającą istotę danego związku. Obok nazwy może wystąpić trójkątny znacznik pokazujący kierunek, w jakim należy odczytywać powiązanie.



# Asocjacja 4: Role

## Role

Klasa uczestnicząca w powiązaniu odgrywa w nim pewną rolę. Rola może być nazwana i pokazana na diagramie.

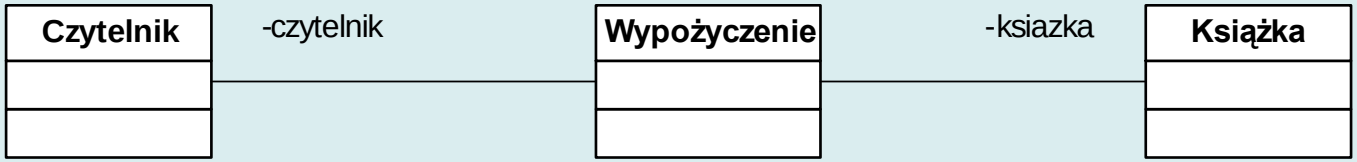



Wiele narzędzi do tworzenia diagramów UML jest w stanie automatycznie wygenerować kod klas na podstawie diagramu. Zazwyczaj nazwa roli jest wtedy wykorzystywana jako nazwa atrybutu w klasie występującej z drugiej strony asocjacji.

# Asocjacja 5: Role

**Przykład**: wygenerowany zostanie kod klasy Wypożyczenie zawierający referencje (Java) lub wskaźniki (C++) o nazwie czytelnik i ksiazka.

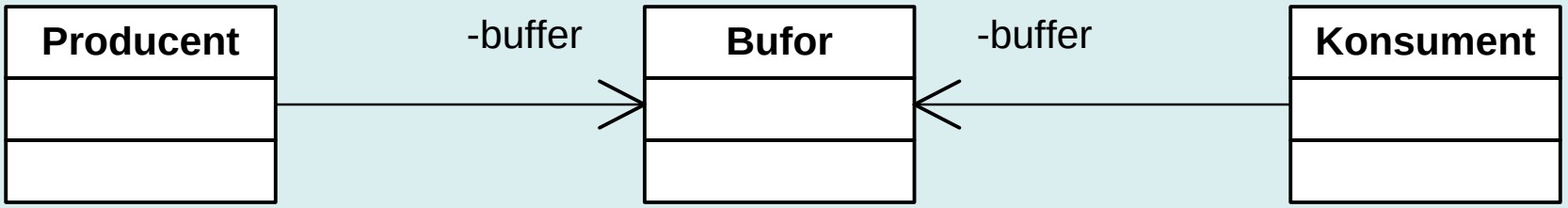
Nie należy wprowadzać do zestawu klasy atrybutów, które są odpowiedzialne za role, bo wystąpiłyby dwukrotnie…




```
class Wypozyczenie
{
  Czytelnik czytelnik;
  Ksiazka ksziazka;
  Wypozyczenie(Czytelnik c, Ksiazka k){
    czytelnik = c;
    ksiazka = k
  }
}
```

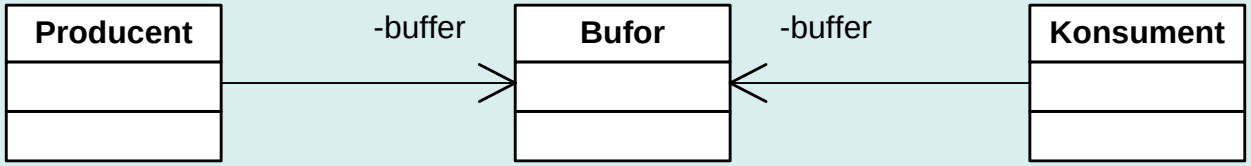
# Asocjacja 6: Nawigacja

Asocjacja jest w zasadzie relacją dwukierunkową, co oznacza, że nawigacja jest możliwa w obie strony. W wielu przypadkach taka nawigacja nie jest wymagana, stąd na diagramie może być wskazany kierunek nawigacji.

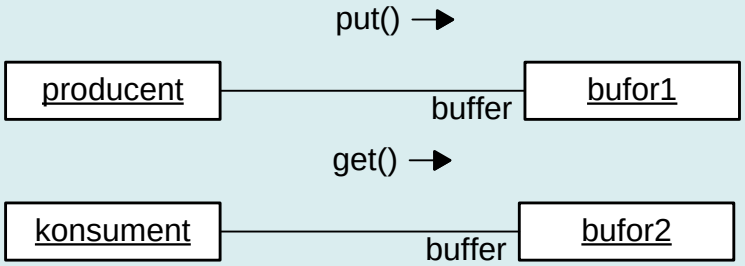



W sytuacji pokazanej na rysunku możliwa jest nawigacja od obiektów klasy Producent i Konsument do obiektu klasy Bufor, ale nie w przeciwną stronę. W klasach Producent i Konsument jest przechowywana informacja o buforze (atrybuty buffer mogą zostać wygenerowane na podstawie nazwy roli asocjacji).

# Asocjacja 7: Nawigacja




Warto zwrócić uwagę, że z diagramu klas nie wynika, że Producent i Konsument dzielą wspólny obiekt klasy Buffer. Jest to widoczne na diagramie obiektów lub komunikacji.




# Asocjacja 8: Liczebność

Liczebność (krotność) asocjacji określa ile obiektów może zostać połączonych przez jeden egzemplarz powiązania. Liczebność może być konkretną wartością lub być określona jako przedział.

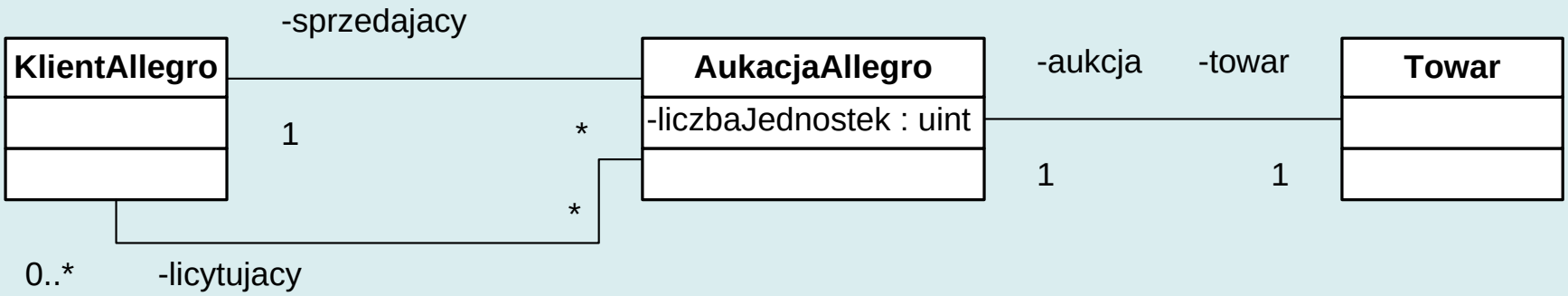
| Oznaczenie | Interpretacja |
|---|---|
| 1 | Dokładnie jeden |
| n | Dokładnie n (n>1) |
| * | Wiele (dowolna liczba) |
| 0..* | Zero lub wiele |
| 0..1 | Zero lub 1 |
| 1..* | Jeden lub wiele |

# Asocjacja 9: Liczebność

Liczebność może dotyczyć każdego elementu biorącego udział w relacji (każdego zakończenia łuku)

# Asocjacja 10: Wielokrotne asocjacje

Pomiędzy klasami mogą występować wielokrotne asocjacje odpowiadające różnym rolom.

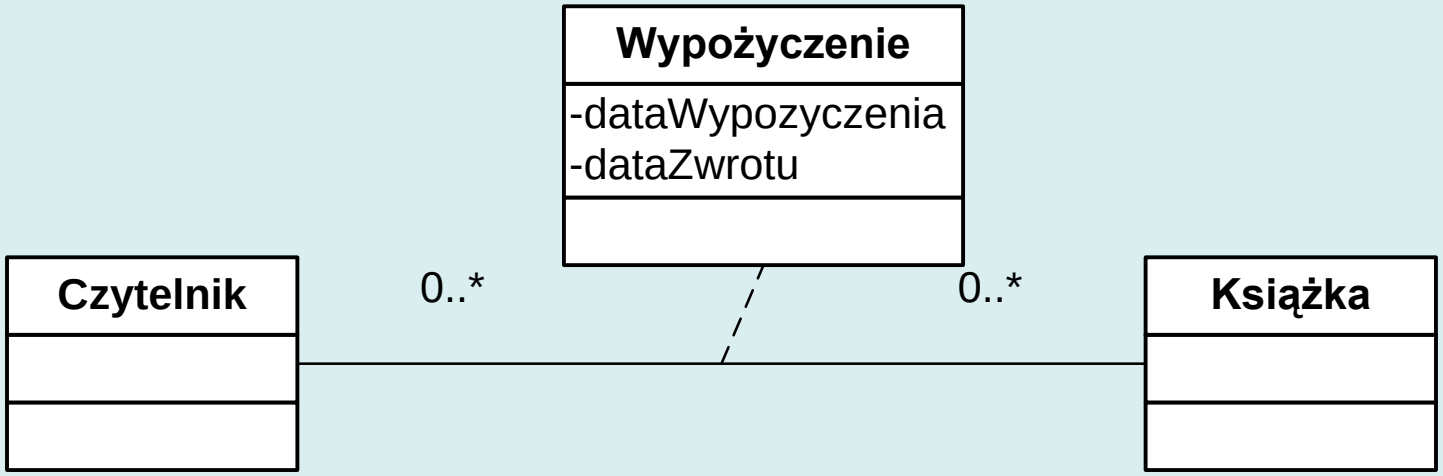



W przedstawionym przykładzie AukcjaAllegro ma dokładnie jednego sprzedającego i 0..* licytujących.

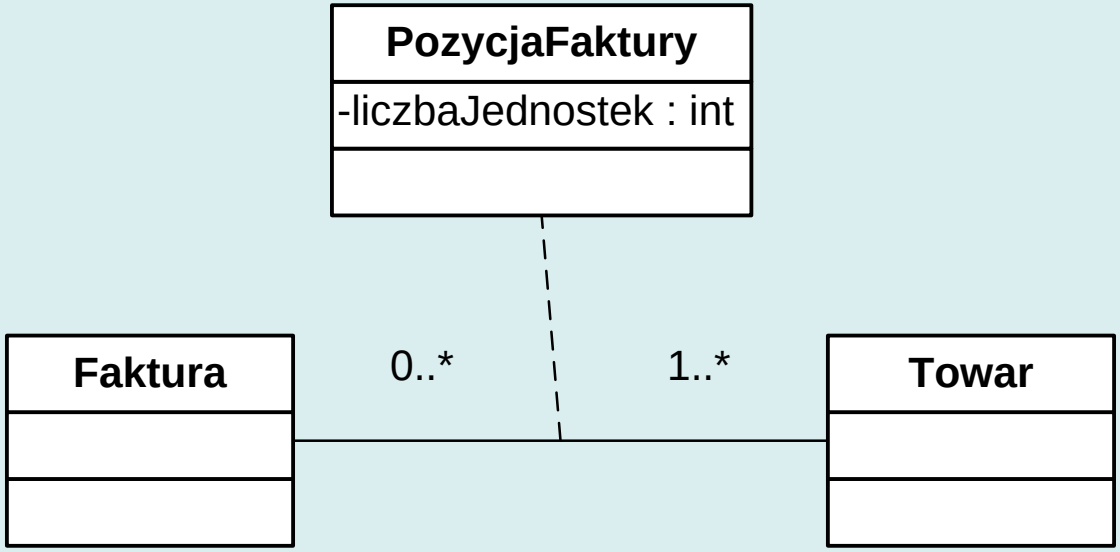
# Asocjacja 11: Asocjacja wiele do wielu

Asocjacja, w której z każdej strony może wystąpić wiele obiektów jest kłopotliwa w implementacji. Do jej realizacji musimy wykorzystać dodatkowe struktury danych (klasy, rekordy bazy danych).

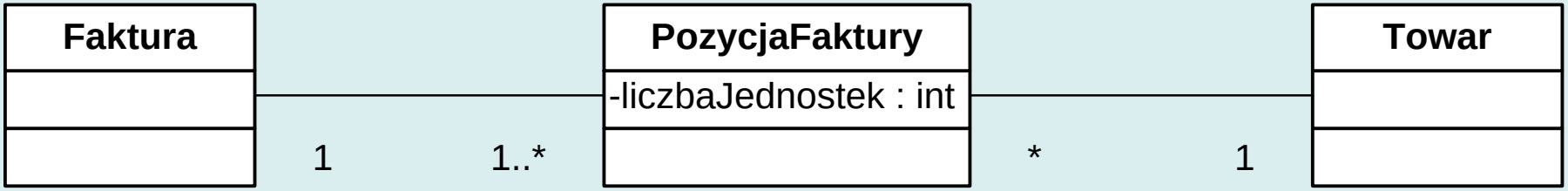
Mogą być one wprowadzone ze względów czysto technicznych, ale często zawierają one dodatkowe atrybuty. W takim przypadku do asocjacji można dowiązać klasę odpowiedzialną za jej realizację (*association class*).



# Asocjacja 12: Asocjacja wiele do wielu




Ta sama relacja może być wyrażona jako:



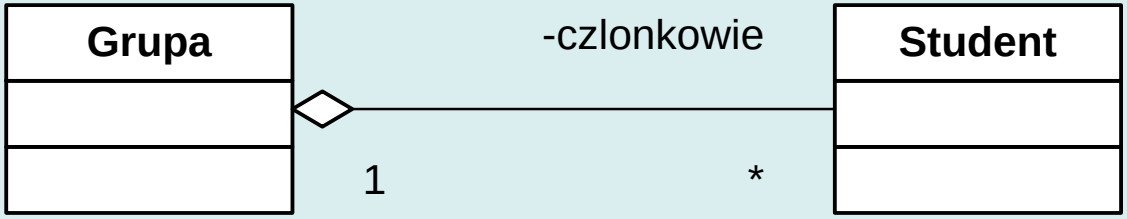
# Agregacja 1

Asocjacja jeden do wielu jest bardzo często identyfikowana jako agregacja, czyli struktura całość-część, gdzie jedna klasa jest częścią większej całości.

Frazą języka naturalnego wskazującą na istnienie agregacji jest *ma*, *posiada*, *zawiera:*

- *Grupa zawiera Studentów*
- *Pojazd ma Koła*
- *Biblioteka posiada Książki*
- *Faktura ma PozycjeFaktury*

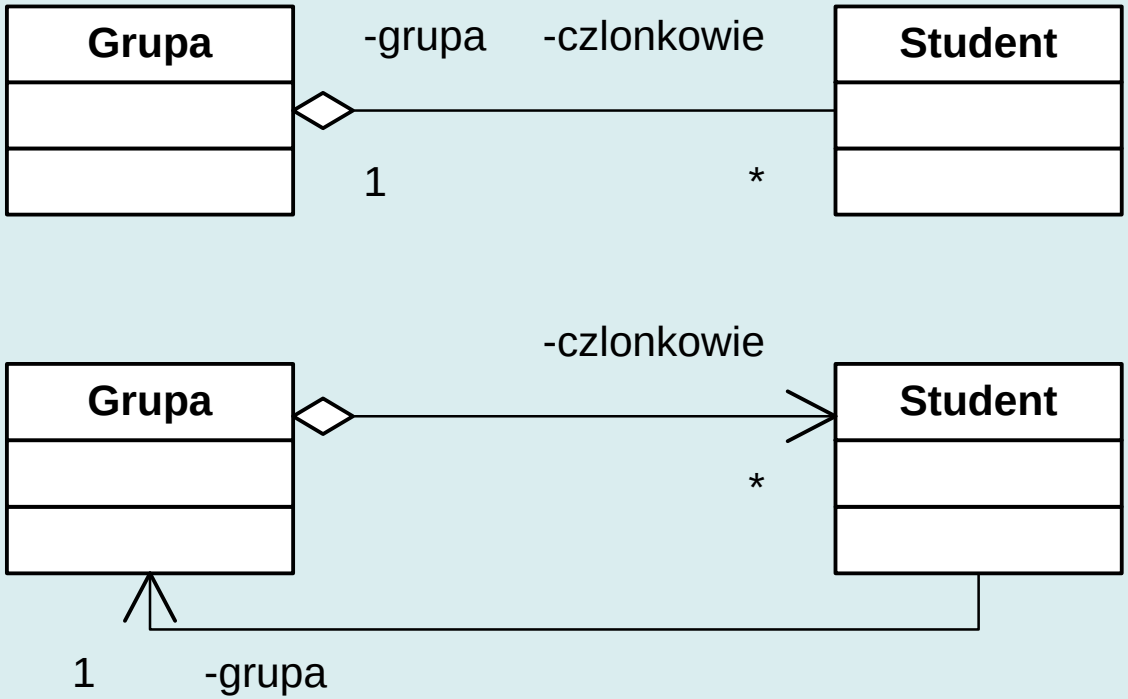
# Agregacja 2




- W zasadzie agregacja nie określa ani kierunku nawigacji, ani też czasu życia obiektów wchodzących w skład struktury.
- Jednakże istnienie agregacji w praktycznych implementacjach oznacza, że obiekt będący posiadaczem (reprezentujący „całość”) używa kontenera do składowania posiadanych elementów (reprezentujących „części”) i w związku z tym standardowo dostępna jest nawigacja od posiadacza do elementu posiadanego.

# Agregacja 3

- Nawigacja w przeciwną stronę może zależeć od praktycznych rozwiązań stosowanych przy generacji kodu i sposobu interpretacji wyspecyfikowanych ról.



# Kompozycja 1

Kompozycja jest szczególnym przypadkiem agregacji. W przypadku kompozycji obiekt reprezentujący całość składa się z części.

- *Województwo składa się z Powiatów*
- *Powiat składa się z Gmin*
- *Rysunek składa się z ElementówRysunku*

# Kompozycja 2

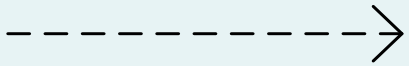
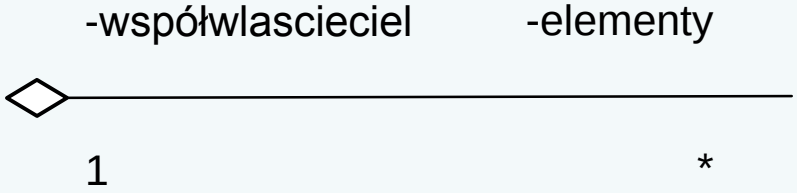
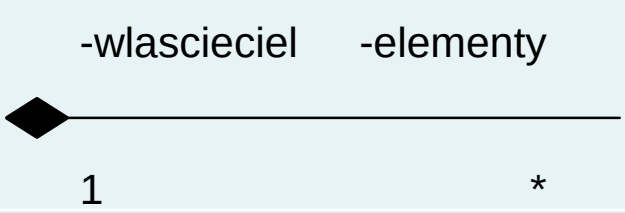
- Potencjalnie, w przypadku agregacji elementy składowe mogą być dzielone, w przypadku kompozycji są one wyłączną własnością elementu nadrzędnego.
- Na ogół element nadrzędny kontroluje czas życia składowych. Wraz z destrukcją całości, są one usuwane.
- UML nie definiuje jednak semantyki. Różnice pomiędzy agregacją i kompozycją są umowne.

# Kompozycja 3

W odniesieniu do systemów informatycznych i języków programowania rozróżnienie pomiędzy agregacją i kompozycją jest uzależnione od platformy, na której będą implementowane klasy występujące na diagramie.

- Język C++ promuje techniki kompozycji, wbudowane mechanizmy zapewniają usuwanie obiektów klas będących atrybutami, ze względu na konieczność jawnego zarządzania pamięcią kontenery są z reguły wyłącznymi właścicielami elementów. Równocześnie agregacja (rozumiana jako współdzielenie elementów) jest możliwa do zaimplementowania, ale jest traktowana jako rozwiązanie potencjalnie zwiększający zagrożenie wystąpienia błędów.
- Z kolei w języku Java (także C#) model zarządzania pamięcią promuje agregację -- czas życia obiektów nie jest dobrze określony i pozostaje pod kontrolą mechanizmu *garbage collection* maszyny wirtualnej.

# Podsumowanie relacji

| Symbol graficzny | Relacja |
|---|---|
|  | Zależność. Klasy wiedzą coś o sobie. |
|  | Dziedziczenie (specjalizacja) |
|  | Asocjacja (powiązanie). Klasy mają atrybuty odpowiedzialne za implementację relacji |
|  | Agregacja. Mocniejsze powiązanie, obiekty mogą być dzielone. |
|  | Kompozycja. Najmocniejsze powiązanie, obiekt nadrzędny składa się z podrzędnych |