Programowanie obiektowe

W paradygmacie obiektowym, modelujemy rzeczywistość przez **obiekty**:

- Obiekt może mieć stan (atrybuty etc.).
- Na obiektach można wykonywać działania (np. wywoływać ich metody).
- Pomiędzy obiektami mogą występować relacje.

W Pythonie (i wielu językach zorientowanych obiektowo): każdy obiekt jest obiektem (instancją) pewnej klasy.

Wiele relacji między obiektami wynika z relacji między ich klasami.

Architektura systemu (informatycznego): (minimalistyczna definicja): komponenty używane przez system wraz z relacjami pomiędzy nimi.

W aplikacji w języku obiektowym: klasy (wraz z atrybutami i interfejsem) i zależności (powiązania) między klasami.

Przykładowo: implementacja grafów z wykładu używa dwóch klas: Graph i Vertex. Definicja klasy Graph „używa" klasy Vertex, np.:

- Atrybut vert_list grafu to słownik, gdzie wartościami są instancje Vertex.
- Metoda add_vertex konstruuje nowe instancje klasy Vertex.
- Metoda add_edge wywołuje metodę add_neighbor na instancjach Vertex.
- ...

Definicja klasy Vertex nie używa klasy Graph.

Diagramy klas

UML (*Unified Modeling Language*) – język służący do opisu architektur.
Szczególne zastosowanie: **diagramy klas**.

Formalizuje pewne intuicyjne sposoby przedstawiania klas i zależności między nimi.

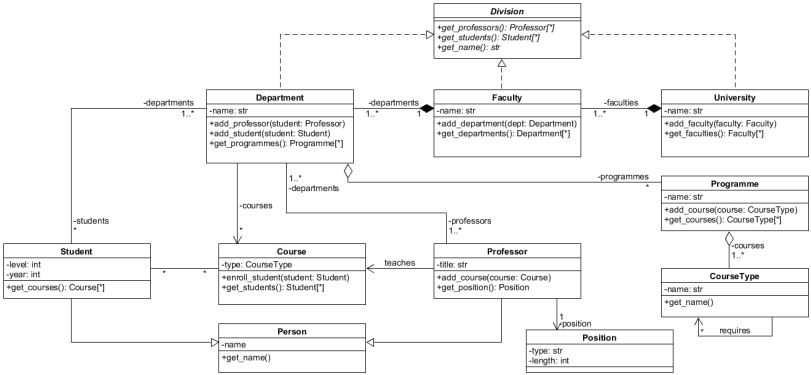
Z jednej strony jest dość ścisły – UML 2.5, standard sankcjonowany przez ISO/IEC 19505-2:2012.

Z drugiej strony jest półformalny: składnia jest elastyczna, podlega konwencjom. Jego odbiorcami są ludzie.

O ile to możliwe, jest niezależny od platformy (języka programowania, etc.).

Dalsze slajdy: **niektóre** elementy języka UML dla diagramów klas.

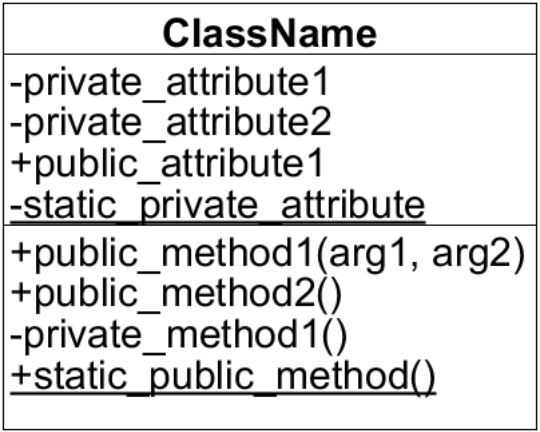
Przykład:



Pudełka reprezentują klasy. Krawędzie reprezentują relacje między **instancjami** danych klas.

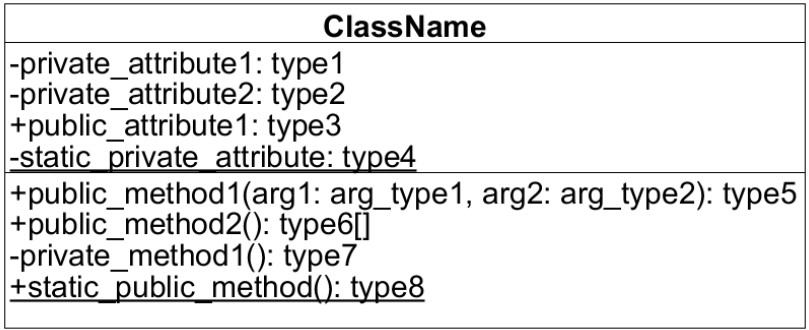
Diagramy UML

Reprezentacja klasy: nazwa, atrybuty, operacje. Dekoracja atrybutów/operacji: prywatne ($-$), publiczne ($+$), statyczne (podkreślenie).

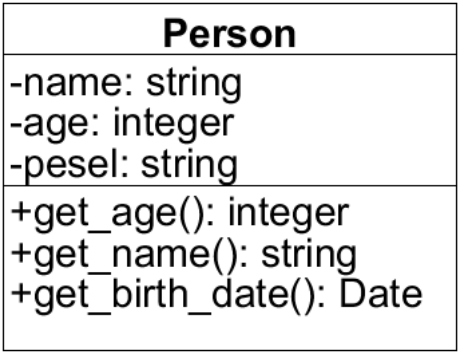


Diagramy UML

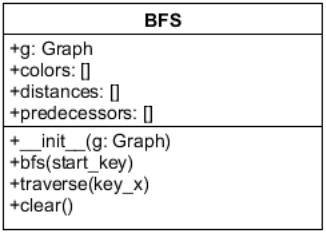
Atrybuty, argumenty operacji i typy obiektów zwracane przez operacje mogą być określonego typu. Kolekcje obiektów (np. listy): $\mathrm{type}[]$.

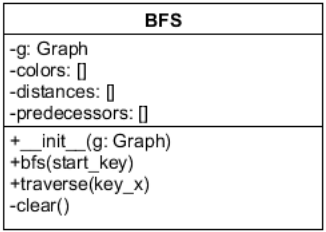


Przykład: Person, klasa reprezentująca osobę. Wszystkie atrybuty prywatne, wszystkie operacje publiczne. Date – inna klasa, reprezentująca datę.



Przykład (bliżej Pythona): BFS, „wszystko publiczne" vs. niektóre elementy ukryte (hermetyzacja). Konwencja dla nazw jak w Pythonie: atrybuty i metody prywatne zaczęte od „_". Kolekcje (listy etc.) oznaczone przez [].





Diagramy UML

Relacje między klasami – podstawowe rodzaje (od najsłabszej do najsilniejszej):

Zależność (dependency)

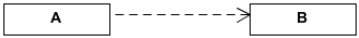
Asocjacja (association)

Agregacja (aggregation)

Kompozycja (composition)

Dziedziczenie (inheritance)

Zależność – najsłabszy rodzaj relacji. Klasa A zależy od klasy B, gdy A wykorzystuje B w działaniu[1]. Zmiany w definicji B mogą wymagać zmian w klasie A (ale niekoniecznie odwrotnie).



Z reguły zależność powstaje, gdy A używa B „przelotnie". Przykładowo:

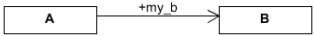
- Argumentem pewnej operacji A jest obiekt typu B; A używa pewnych operacji B.
- Pewna operacja A tworzy i zwraca obiekt typu B.

Takie zależności można opisać jako adnotacje w diagramie:

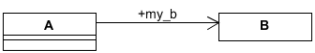
[1]Tu mamy skrót myślowy: *instancje* A wykorzystują *instancje* B w działaniu.

Asocjacja – silniejsza zależność między klasami.

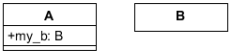
Zapis jest równoważny stwierdzeniu, że pewien atrybut (obiektu) klasy A przechowuje obiekt(y) typu B. A i B są wtedy związane na dłuższy czas (A „przechowuje”, „posiada” B), jednak czasy istnienia instancji A i B mogą być od siebie niezależne.
Przykład: $\mathrm{Uniwersytet}$ i Osoba: uniwersytet „pamięta” osoby, które na nim studiują. Osoba może studiować na wielu uniwersytetach jednocześnie. Osoba nie przestaje istnieć, gdy przestaje studiować.
Nazwa atrybutu A przechowującego B może się pojawić jako adnotacja.



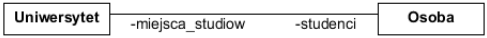
Asocjacje stosujemy zamiennie z atrybutem:



lub



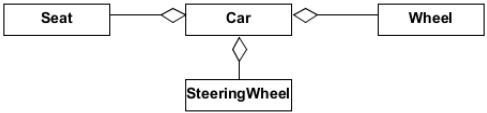
Asocjacja obustronna: gdy (instancje) A „pamiętają” B i odwrotnie.
Przykład: $\mathrm{Uniwersytet}$ przechowuje informację o osobach, które na nim studiują, ale Osoba może też przechowywać informację, na jakim uniwersytecie (uniwersytetach) studiuje.



Agregacja (agregacja częściowa) – obiekt typu B jest częścią składową obiektu typu A.

Obiekt typu A nie musi mieć wyłączności: ten sam obiekt typu B może być częścią wielu obiektów typu A, a jego istnienie nie jest uzależnione od istnienia (konkretnego) obiektu A.

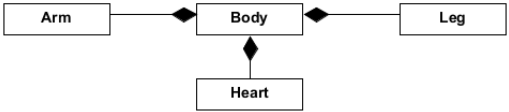
Przykład: Samochód składa się m.in. z kół, siedzeń i kierownicy. Każde koło etc. w trakcie istnienia mogą być częścią różnych samochodów.



Kompozycja (agregacja pełna) – agregacja, w której istnienie obiektu typu B zależy w całości od istnienia obiektu typu A.

Z reguły obiekt typu A jest wtedy odpowiedzialny za tworzenie i niszczenie obiektów typu B wchodzących w jego skład. Usunięcie obiektu typu A oznacza zniszczenie jego składowych obiektów typu B.

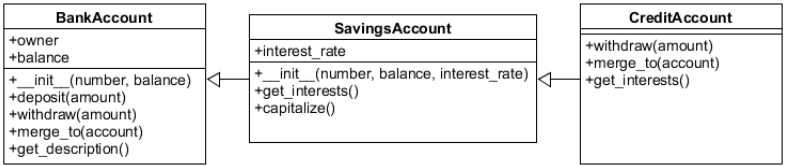
Przykład: Składowymi ciała są m.in. ręce, nogi i serce. Składowe nie mogą (pomijając cuda współczesnej medycyny) istnieć bez ciała.



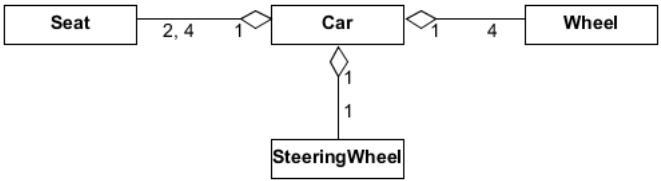
Dziedziczenie – klasa A jest podklasą B.

Podklasy dziedziczą atrybuty i operacje (jest to relacja przechodnia). Mogą jednak nadpisywać ich zachowanie.

Przykład: konta bankowe.



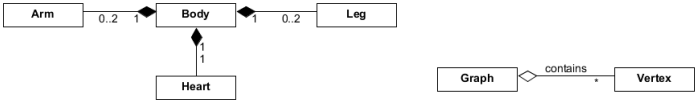
Krotność relacji. Niektóre relacje asocjacji wiążą wiele obiektów obu klas (osoba może studiować na wielu uniwersytetach, uniwersytet ma wielu studentów), lub po jednym obiekcie (jedno ciało ma jedno serce i odwrotnie), lub mogą być relacją wiele-do-jednego/jeden-do-wielu (samochód ma cztery koła, a koło w danym momencie jeden samochód).
Krotność reprezentujemy symbolicznie jako adnotacja na diagramie, np.:



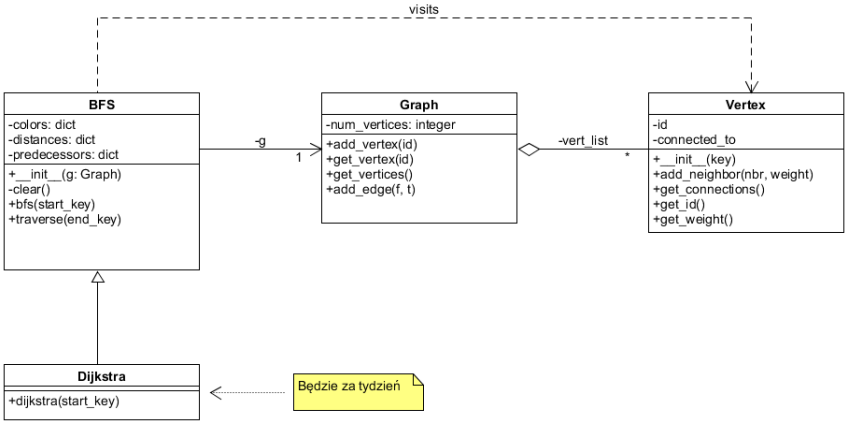
Diagramy UML

Używane oznaczenia krotności:

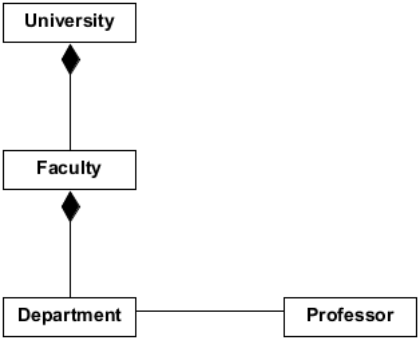
1. $\mathrm{a}..\mathrm{b}$ – od a do b obiektów.
2. $0..1$ – zero lub jeden obiekt.
3. $*$ – dowolna ilość obiektów.
4. $1..*$ – co najmniej jeden obiekt.
5. n_1, n_1, $\ldots$, n_k – konkretne możliwości na ilość obiektów.



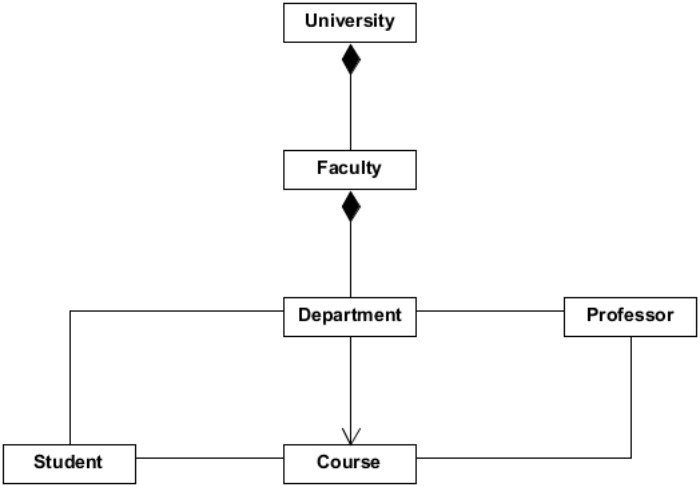
Przykłady diagramów UML



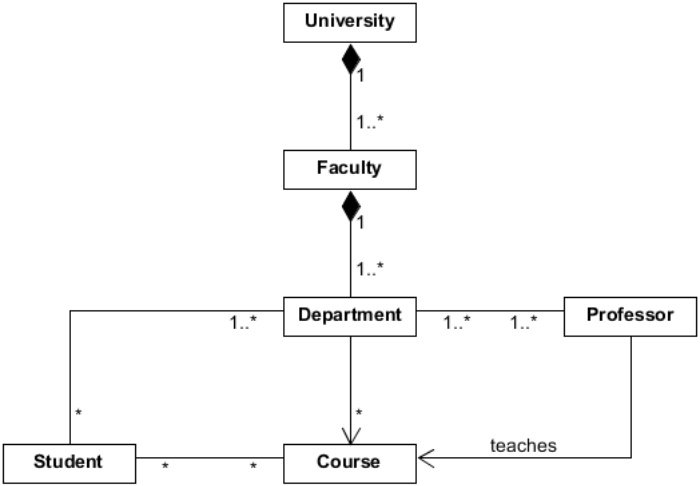
Przykłady diagramów UML



Przykłady diagramów UML



Przykłady diagramów UML



Przykłady diagramów UML