Modelowanie obiektowe

Janusz Jabłonowski

9 października 2022

[1] Na podstawie slajdów o modelowaniu autorstwa Jacka Sroki

Wprowadzenie do programowania obiektowego

- *Programowanie obiektowe* (object-oriented programming).
- Program = zbiór komunikujących się obiektów
- *Obiekt* ma stan i zachowanie
- Obiekty o tym samym zachowaniu i takich samych elementach stanu grupujemy w *klasy*

Historia programowania obiektowego

- Simula-67[1] Ole-Johan Dahl, Kristen Nygaard, Oslo
- Symulacja ruchu statków
- Obiekty, klasy, dziedziczenie
- Smalltalk (zespół pod kierownictwem Alana Kaya w Xerox Palo Alto Research Center - PARC)
- Pierwsza wersja Smalltalk-71, pierwsza dostępna poza PARC (i istotnie zmieniona): Smalltalk-80
- Gwałtowny wzrost popularności programowania obiektowego to lata 80-te, głównie za sprawą (hybrydowego) C++
- Obecnie: obiektowość jest wszędzie :)

[1] Konkurs bez nagród - w którym roku powstała Simula-67?

Zalety programowania obiektowego

- Program jest opisem rzeczywistości, używa pojęć wziętych z modelowanego świata
- Zmniejszenie luki reprezentacji (ang. representational gap)
- Jest oparte na pomysłach znanych z codzienności

Fazy tworzenia oprogramowania[2]

- Analiza
- Projektowanie
- Implementacja
- ...
- W podejściu obiektowym te fazy nie dają się bardzo wyraźnie rozdzielić (to zaleta)
- Wspólny język wszystkich faz

[2] Najprostszy model

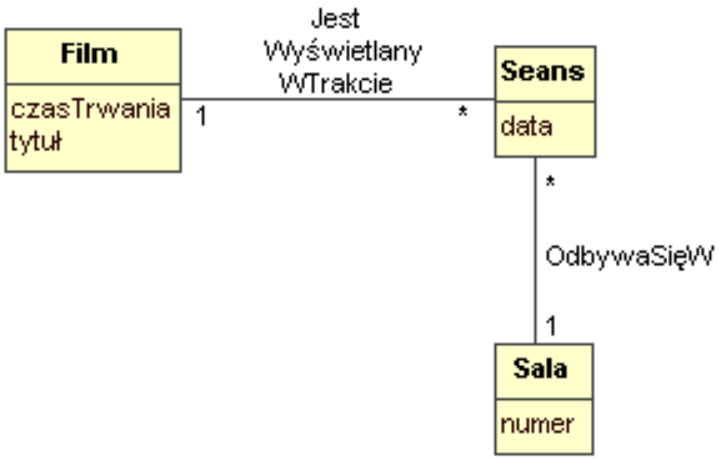
Metodologie tworzenia oprogramowania

- Model kaskadowy
- Model iteracyjny
- Programowanie przyrostowe
- Iteracje (2 do 6 tygodni)
- Więcej: Inżynieria Oprogramowania

Analiza i projektowanie obiektowe

- Analiza i projektowanie obiektowe (ang. Object-Oriented Analysis and Design, OOA/D)
- Obiekty pojęciowe - reprezentują pojęcia i koncepcje ze świata rzeczywistego
- Model dziedziny:
  - klasy pojęciowe (ang. conceptual class), czyli pojęcia
  - powiązania między klasami pojęciowymi
  - atrybuty klas pojęciowych

Przykład modelu dziedzinowego



- Rezerwacja miejsc i sprzedaż biletów kinowych
- Nazwy klas - rzeczowniki, nazwy operacji czasowniki
- Kierunek czytania - góra-dół, lewo-prawo, lewe strony linii
- Skąd wziąć te nazwy?
- Ba ...
- Reguła spod kciuka (rzeczowniki i czasowniki z opisu)
- No wiem ...[3]

[3] Ale gdyby było łatwo, to analitycy stracili by pracę ... .

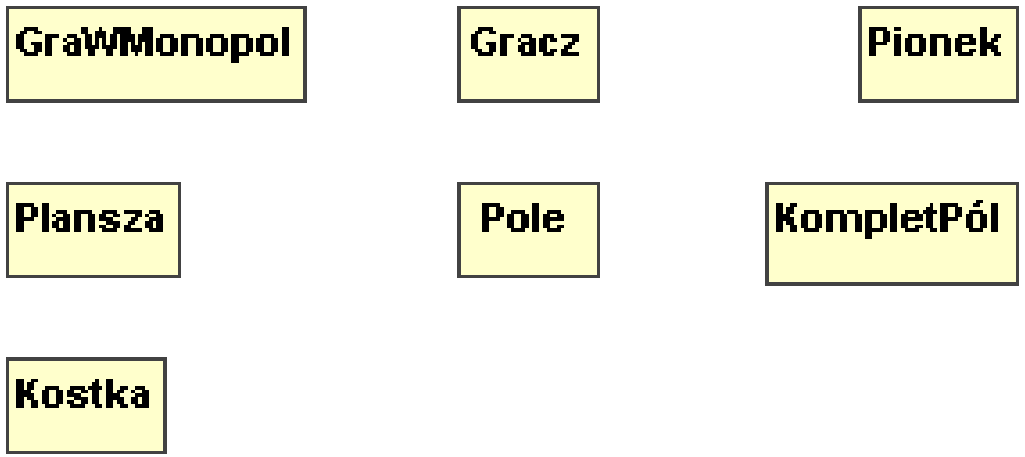
Wskazówki do szukania klas pojęciowych

- Istniejące nazwy
- Tylko dotyczące modelu

Wskazówki do szukania klas pojęciowych cd

- Często spotykane kategorie (p. tabela w ważniaku)
  - transakcja (np. SprzedażBiletu)
  - produkt transakcji (np. Bilet)
  - miejsce transakcji (np. Kasa)
  - role ludzi (np. Kasjer)
  - zdarzenia (np. Seans)
  - obiekty fizyczne (np. Kino)
  - ...

Przykład klas pojęciowych dla Monopolu



- Klasy pojęciowe dla gry w Monopol

Wskazówki do szukania powiązań

- Powiązanie (ang. association)
- Między klasami
- Wskazuje, że między egzemplarzami tych klas może występować jakaś zależność

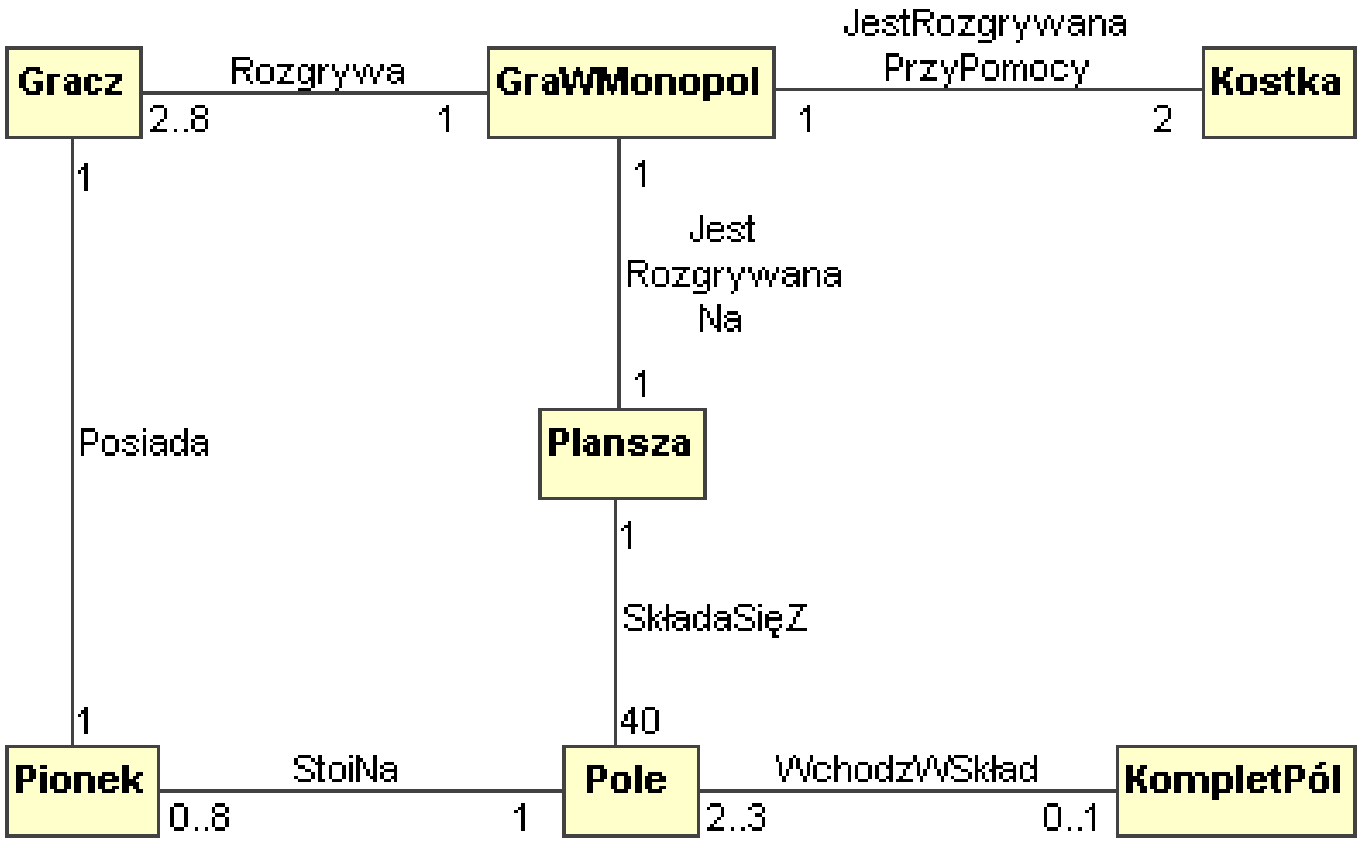
Wskazówki do szukania powiązań cd

- Często spotykane kategorie (p. tabela w ważniaku)
  - A jest fizyczną lub logiczną częścią B (np. Miejsce-Sala Kinowa)
  - A jest uczestnikiem/pracownikiem/członkiem B (np. Kasjer-Kino)

Liczebności powiązań

- Liczebność (ang. multiplicity) określa, jak wiele egzemplarzy klasy A może być powiązane z jednym egzemplarzem klasy B
- Na diagramach przedstawia się w postaci wyrażenia umieszczanego obok klasy A tuż przy linii obrazującej powiązanie
- Przykłady
  - 1(dokładnie jeden)
  - 11 (dokładnie jedenaście)
  - $3, 5, 7$ (trzy lub pięć lub siedem)
  - 2..8 (od dwóch do ośmiu)
  - 0..1 (zero lub jeden)
  - $1..*$ (co najmniej jeden)
  - $*$ (dowolnie dużo, także: zero)

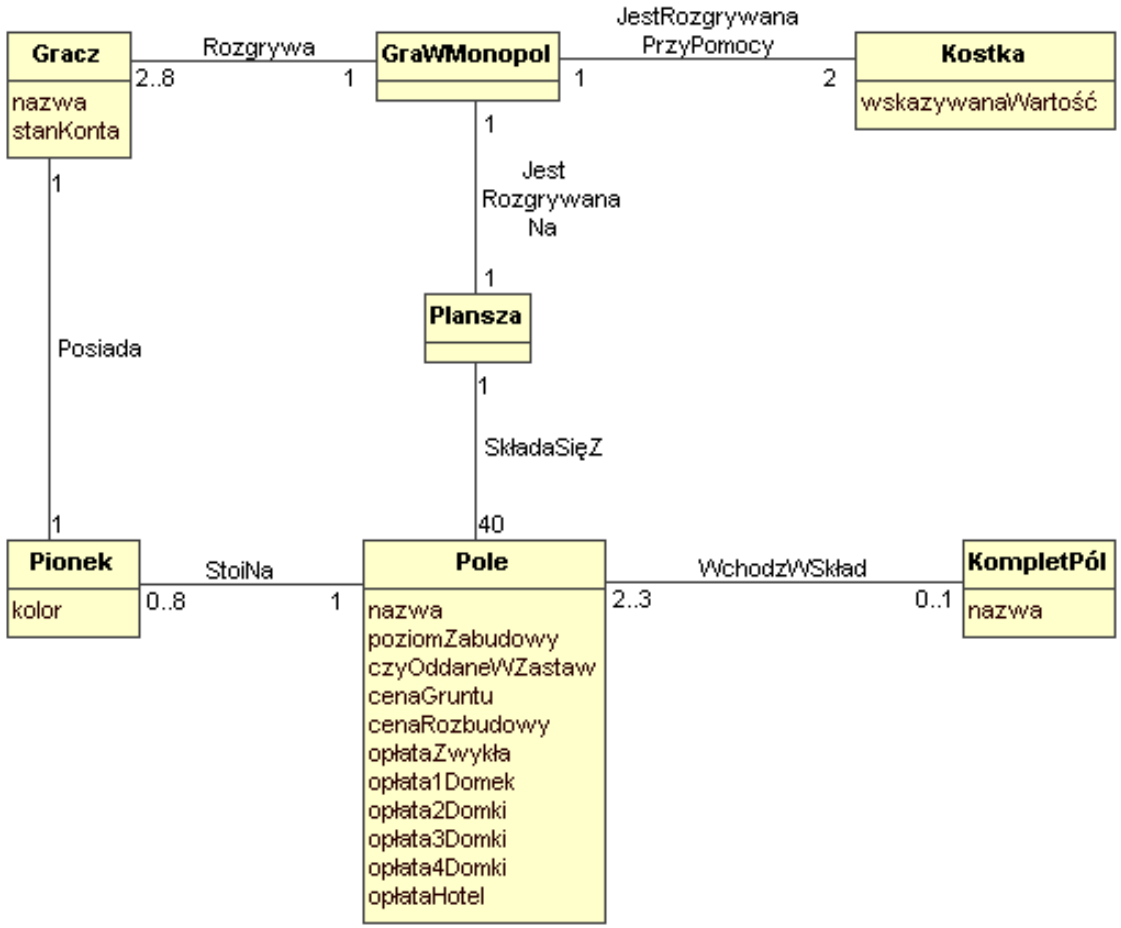
Klasy pojęciowe i powiązania dla gry w Monopol



Atrybuty

- Atrybut (ang. attribute) opisuje egzemplarz klasy pojęciowej
- Problem: czy to atrybut, czy klasa pojęciowa?
- W zasadzie wartości atrybutów powinny pochodzić z typów prostych

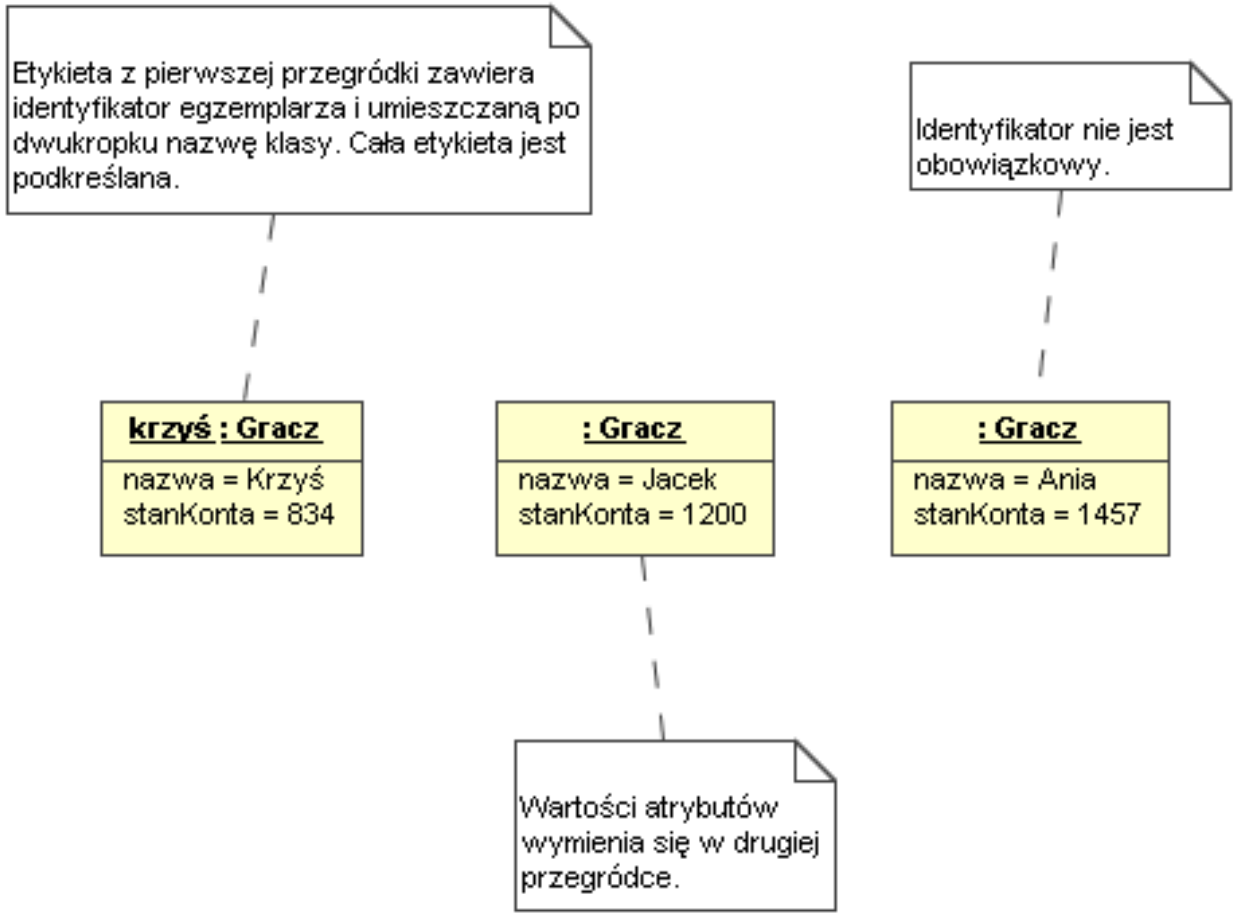
Klasy pojęciowe, powiązania i atrybuty dla gry w Monopol



To nie jest pełny model dla Monopolu

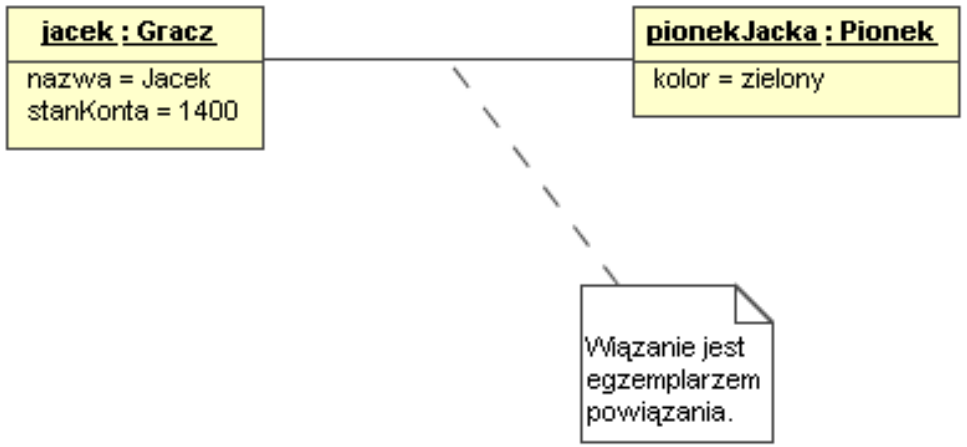
Obiekty i ich stan

Czasem chcemy modelować pojedyncze obiekty oraz notki



Wiązania

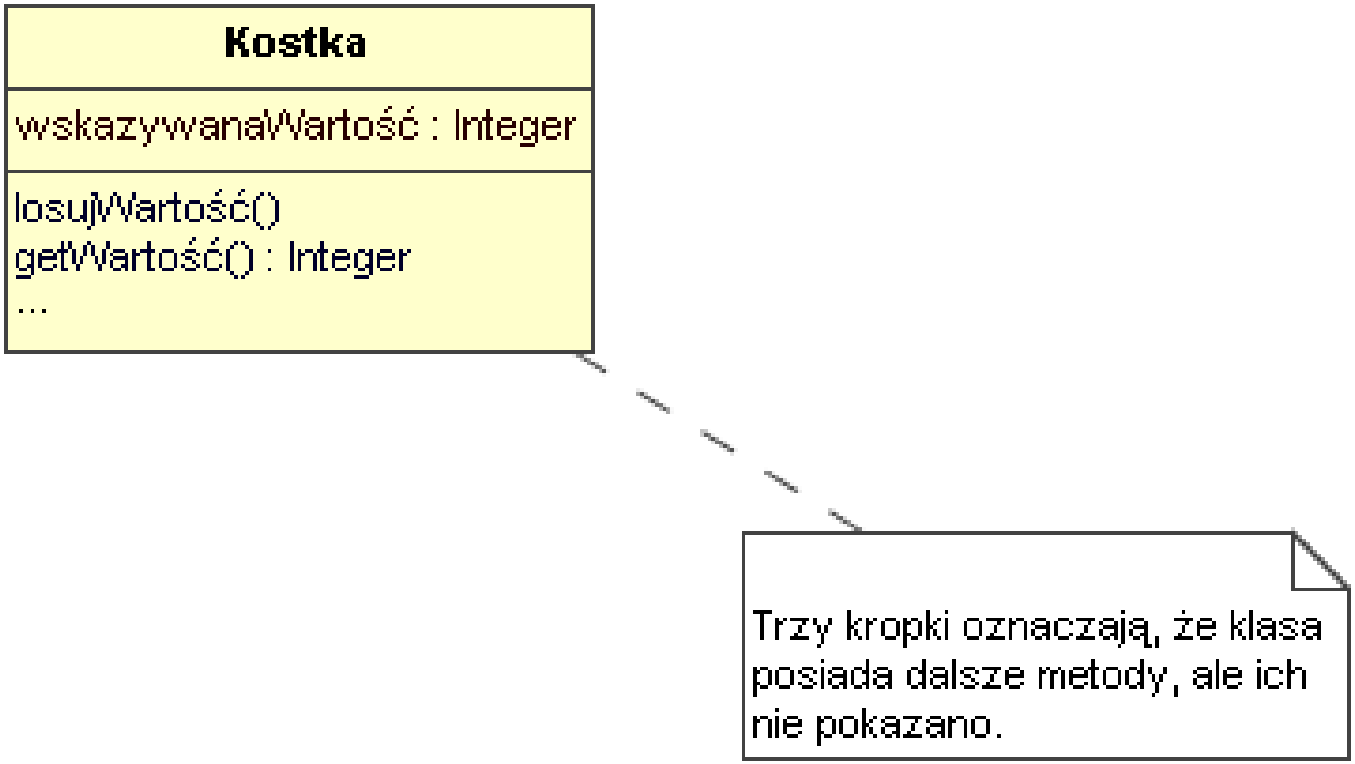
- Wiązanie (ang. link) godny uwagi związek pomiędzy obiektami



Projektowanie obiektowe

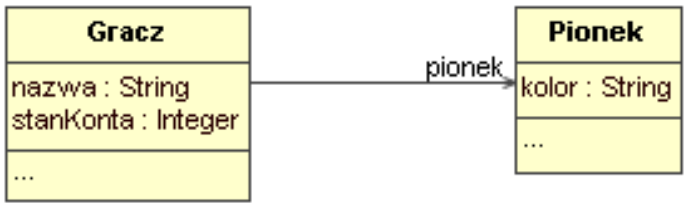
- Przechodzimy od modelu dziedzinowego do projektu
- Przekształcamy klasy pojęciowych w klasy projektowe (ang. design class)
- Obiekty potrafią wykonywać akcje na swoim stanie - metody (ang. method)
- Atrybuty i metody jednej klasy nazywa się jej składowymi (ang. class member)
- Ustalamy typy atrybutów i metod
- Powiązania modelujemy poprzez atrybuty

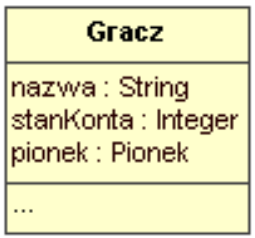
Przykład klasy projektowej

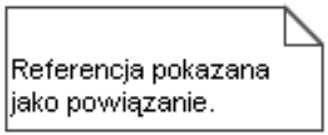


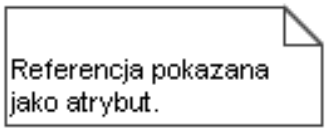
Przykład reprezentowania powiązań

- Jako referencja (ang. reference)
- Jako atrybut
- Powiązania jeden-do-wielu na etapie projektu muszą być reprezentowane jako atrybuty









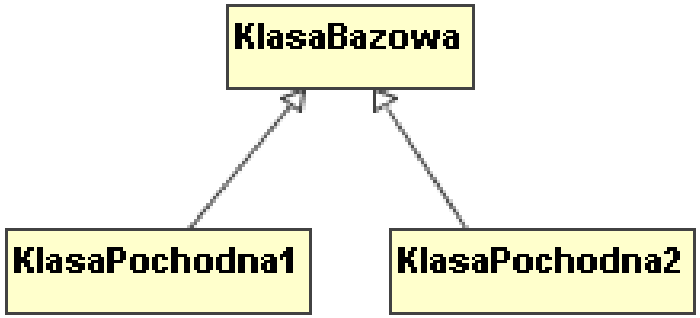
Typy podstawowe

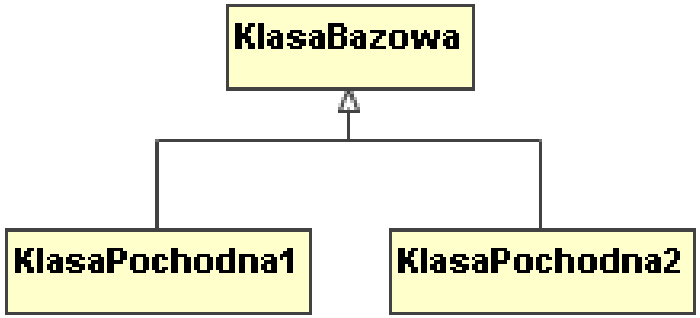
- Typy podstawowe lub proste (ang. primitive type lub basic type).
- Często nie są obiektami (zależy od języka, w Javie niektóre nie są)

Hierarchie klas

- Hierarchie klas (ang. class hierarchy)
- Różne warianty jednego pojęcia (klasy)
- Terminologia: klasa bazowa (ang. base class) bądź nadklasa (ang. superclass), klasa pochodna (ang. derived class) bądź podklasa (ang. subclass)
- Związek między nadklasą i podklasami nazywany jest uogólnieniem (ang. generalization), bądź związkiem uogólnienie-uszczegółowienie.
- Test: czy podklasa jest szczególnym rodzajem nadklasy (czy każdy obiekt podklasy jest obiektem nadklasy)
- Na diagramach uogólnienie pokazywane jest przy pomocy ciągłej linii łączącej klasę bazową i klasę pochodną, która od strony klasy bazowej zakończona jest niewypełnioną trójkątną strzałką.

Hierarchie klas - przykład notacji





Dwa równoważne sposoby pokazywania hierarchii klas.

Kontrakty i widoczność

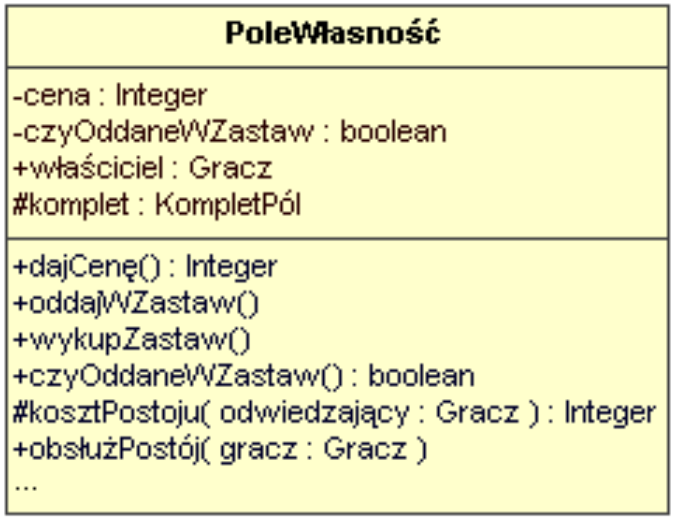
- Projektowanie obiektowe - rozdzielanie odpowiedzialności
- Odpowiedzialność wymaga ochrony danych
- Zakresy widoczności (ang. visibility)
- Składowe mogą być[4]:
  - publiczne (ang. public) – wszystkie klasy,
  - chronione (ang. protected) – ta klasa i jej podklasy
  - prywatne (ang. private) – tylko ta klasa
- Kapsułkowanie (hermatyzacja, ang. encapsulation)

[4] Uwaga: widoczności w Javie są nieco inaczej zdefiniowane (chroniona) i jest jeszcze czwarta kategoria widoczności

Notacja dla widoczności

Notacja

- + publiczna
- # chroniona
- – prywatna



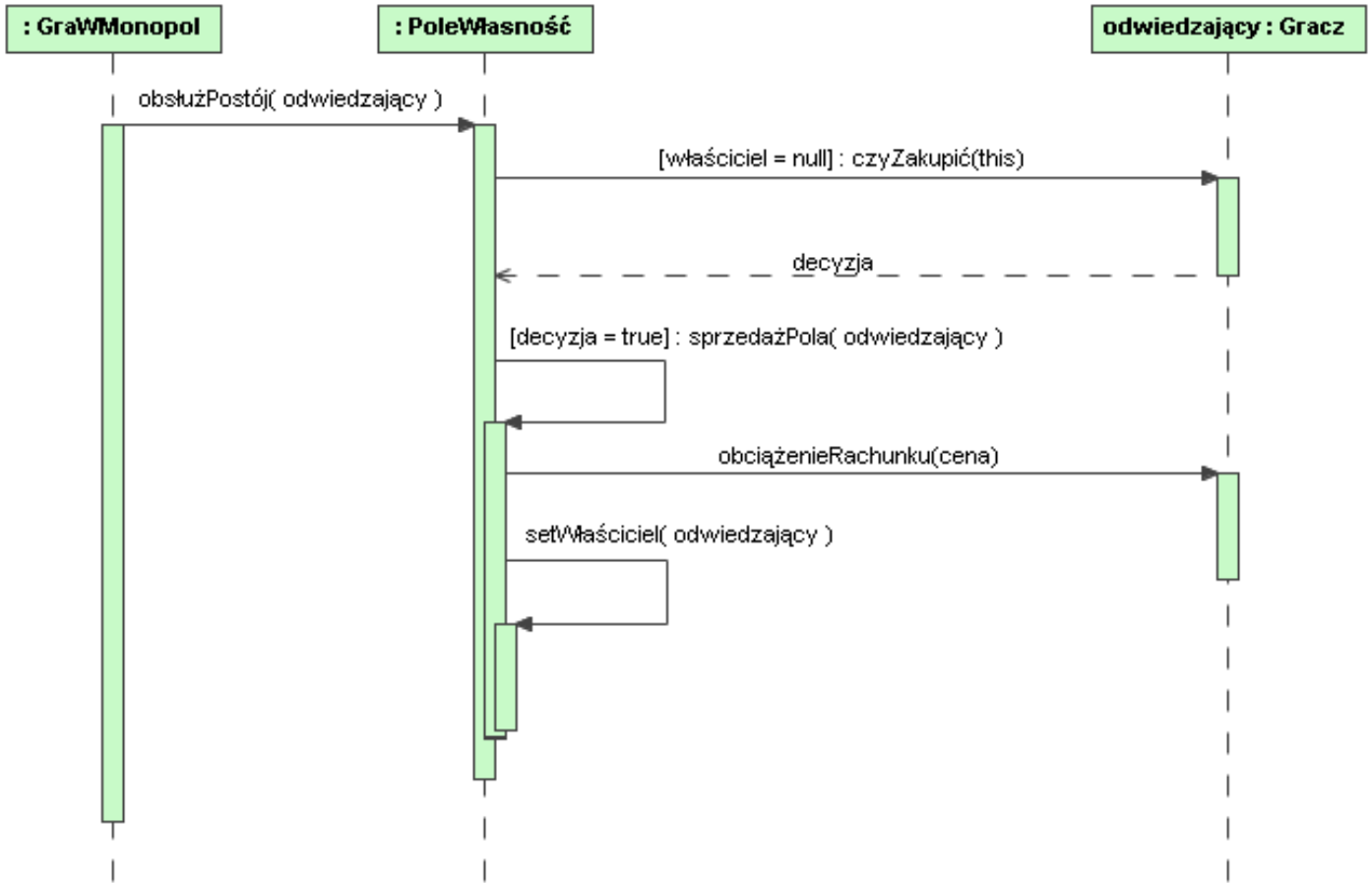
Akcesory

- Akcesory - metody dające dostęp do danych
- Odczytujący (ang. getter), zapisujący (ang. setter)
- W prostych akcesorach jest tylko dostęp do danych
- Akcesory złożone mogą badać poprawność operacji
- Akcesory nie zawierają innej logiki aplikacji
- Nazwy: get*, set* i is*
- Konwencja na tym wykładzie: nazwa atrybutu
- Komplet akcesorów to **nie** jest dobry pomysł!
- Zalety akcesora względem bezpośredniego dostępu do atrybutu
  - może go nie być :)
  - kontrola dostępu
  - ukrywa wewnętrzną implementację

Rozdzielanie zobowiązań

- Trudne
- Czyli ciekawe!
- Pomaga opisanie interakcji między obiektami
- Diagramy przebiegu

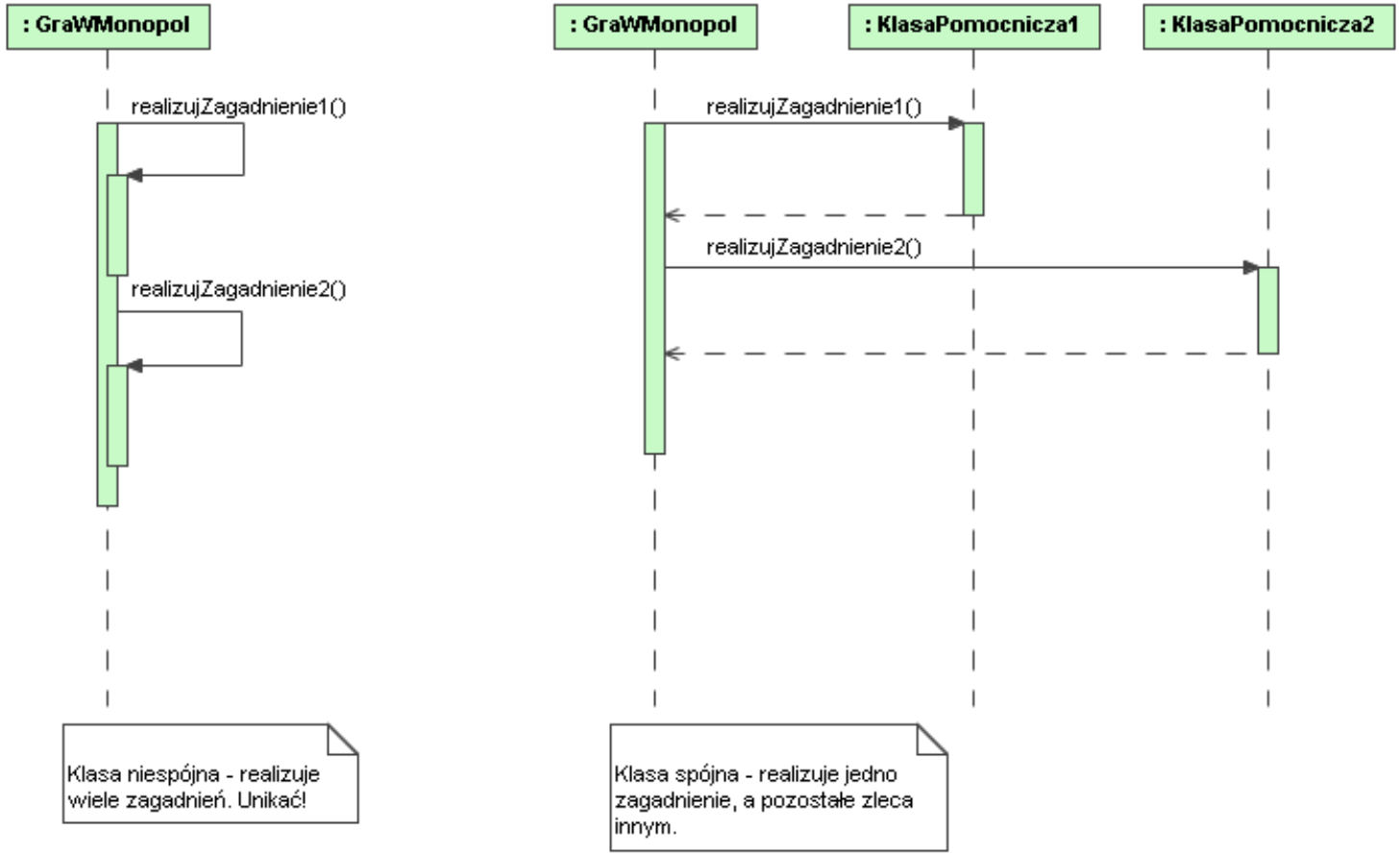
Przykład diagramu przebiegu



Cechy dobrego projektu

- Luźne sprzężenie (ang. loose coupling)
  - ułatwia modyfikowanie, bo im mniej klasy są powiązane ze sobą, tym większa szansa, że zmiany nie będą się propagować
  - wyjątek: nawet silne sprzężenie z biblioteką standardową nie jest groźne
- Wysoka spójność (ang. high cohesion)
  - klasa powinna odpowiadać ze jedno zagadnienie, a nie wiele różnych
  - metody klasy powinny być powiązane ze wszystkimi atrybutami

Przykład niskiej i wysokiej spójności



Kilka ważnych pojęć

- Wzorce projektowe
  - o ile zdążymy, to poświęcimy im cały wykład
- Unified Modelling Language (UML)
  - używamy tylko małego podzbioru tej notacji
  - diagramy klas