Modelowanie i Programowanie Obiektowe

POLITECHNIKA
POZNAŃSKA

Wykład II: Analiza i projektowanie obiektowe

28 październik 2013

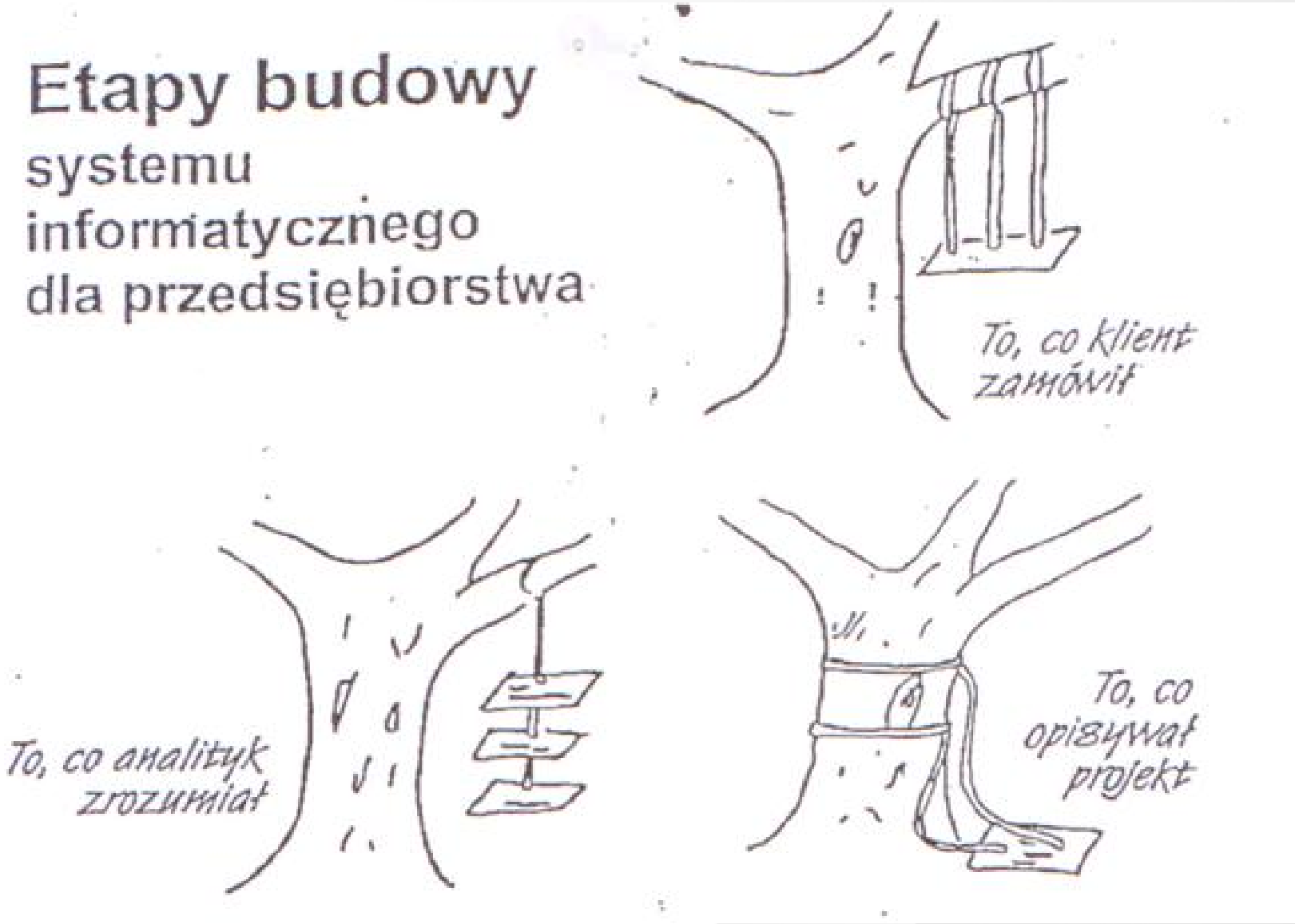
Plan wykładu

- Wstęp do modelowania
  - Motywacja
- Analiza i projektowanie obiektowe
- Modelowanie dziedziny
- Przykładowy stan programu

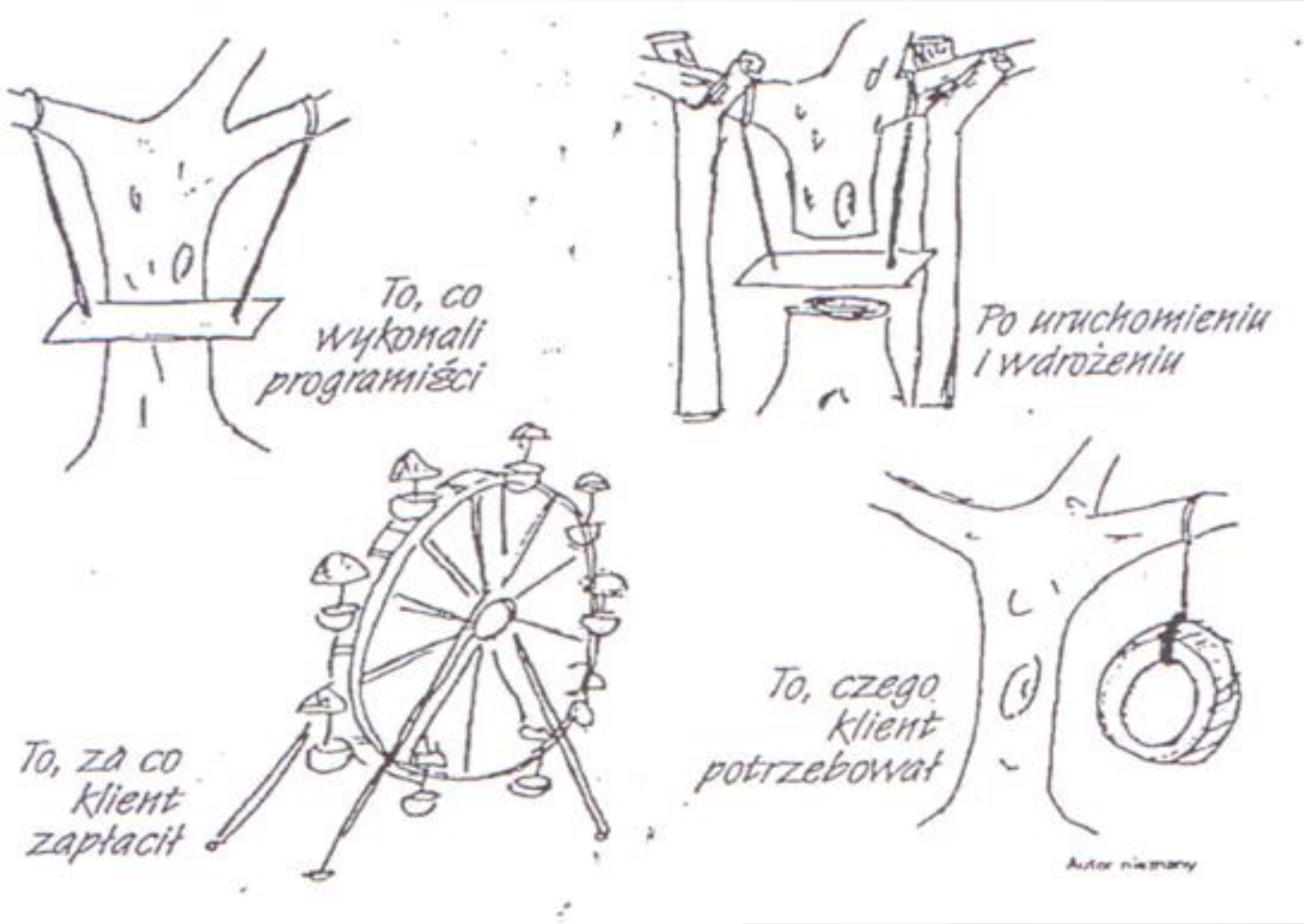
Kolejne kroki budowy systemów informatycznych

- Specyfikacja wymagań
- Analiza dziedziny – modelowanie podstawowych pojęć i terminów
- Analiza systemowa – modelowanie problemu
- Projektowanie – modelowanie rozwiązania
- Programowanie – budowa modelu działającego na komputerze
- Testowanie
- Integracja
- Wdrożenie
- Utrzymanie

Problem konstrukcji złożonych systemów informatycznych



Problem konstrukcji złożonych systemów informatycznych



Przypomnienie

- **Programowanie obiektowe** (*ang. object-oriented programming*) - pogram komputerowy wyrażony poprzez zbiór obiektów będących bytami łączącymi stan (dane) i zachowanie (metody, które operują na danych obiektu). W celu realizacji zadania obiekty wywołują nawzajem swoje metody, zlecając w ten sposób innym obiektom odpowiedzialność za pewne czynności.

Analiza i projektowanie obiektowe

Myśl przewodnia:

"zrób co należy (analiza) oraz zrób to jak należy (projektowanie)"

- **Analiza** to badanie problemu i wymagań, ale nie rozwiązania.
- **Analiza obiektowa** (*ang. object-oriented analisys*) zajmuje się badaniem i klasyfikacją obiektów pojęciowych
- **Obiekty pojęciowe** nie mają nic wspólnego z programowaniem - reprezentują pojęcia i koncepcje ze świata rzeczywistego, a dokładniej z dziedziny, która jest analizowana.

  Przykład:
  Dla systemu Zakładu Transportu Miejskiego obiektami

Projektowanie / projektowanie obiektowe

- **Projektowanie** – koncepcyjne rozwiązanie (programistyczne lub sprzętowe) realizujące wymagania
  - Bez szczegółów implementacji
  - Pomysły i idea są najważniejsze
  - np. opis schematu baz danych lub klas programowych
- **Projektowanie obiektowe** – to projektowanie obiektów programowych
  - Wyznaczenie, który obiekt za co odpowiada
  - Opisanie jak obiekty mają współpracować by wykonać określony cel
  - tzw. *wzorce projektowe* – jak rozwiązać określony problem obiektowo

Proces wytwarzania oprogramowania

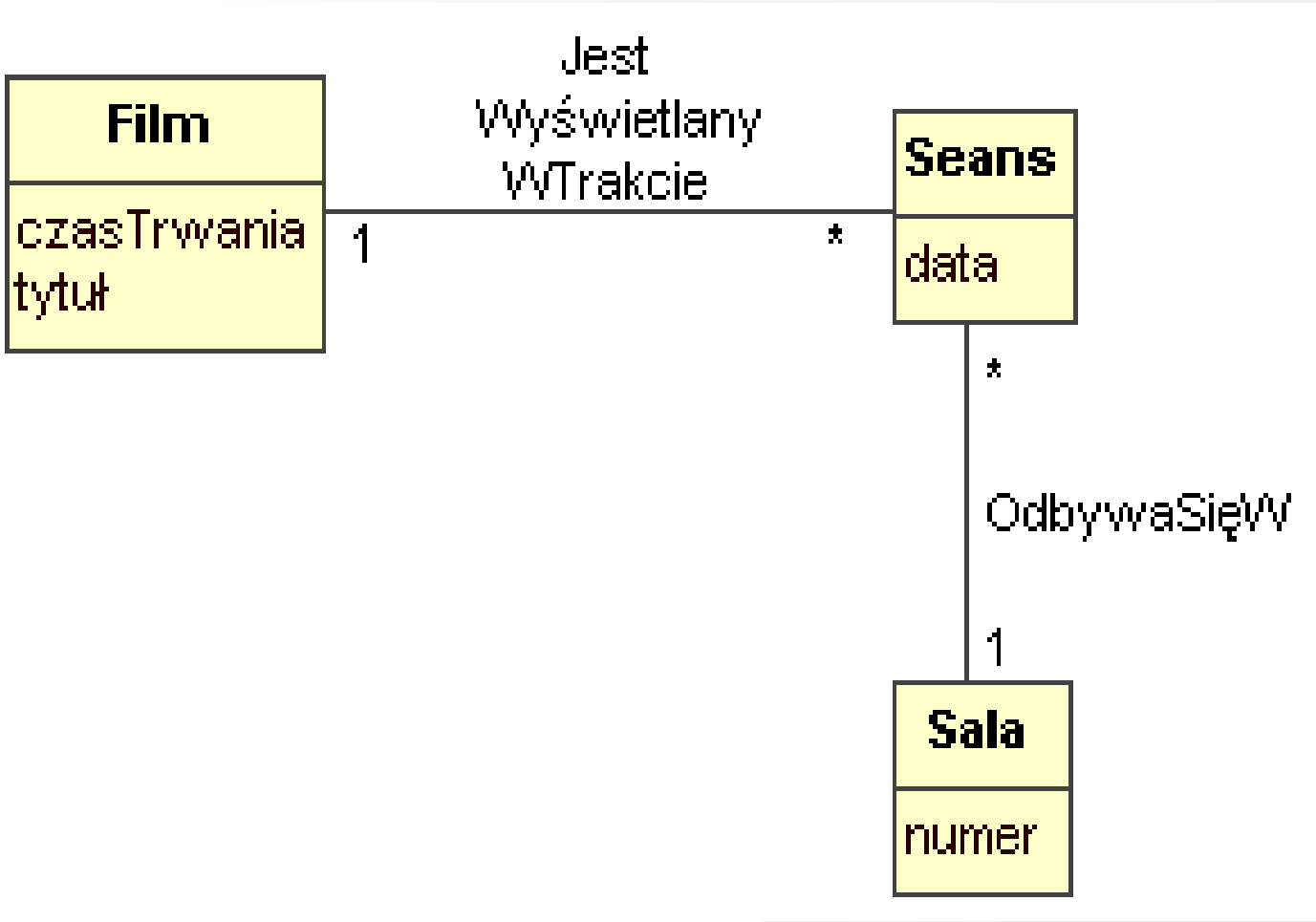
- **Podejście praktykowane:**
  - Przeprowadzenie analizy
  - Opracowanie projektu
  - Testy
  - Wdrożenie
- **Jak byś powinno:**
  - Założenie: *„klient nie wie czego chce”*
  - Tworzenie aplikacji w sposób przyrostowy
    - 2-6 tyg. iteracje

Obiektowe modelowanie dziedziny

- **Model dziedziny** odzwierciedla pojęcia z modelowanej części świata rzeczywistego i ich zależności. W modelu dziedziny pokazuje się:
  - Klasy pojęciowe
  - Powiązania pomiędzy klasami pojęciowymi
  - Atrybuty klas pojęciowych
- Model dziedziny ułatwia jej zrozumienie
- Przy niewielkich projektach może być zbędny

Obiektowe modelowanie dziedziny – diagramy klas

- Model dziedziny przykład:



Obiektowe modelowanie dziedziny

- Każda klasa pojęciowa reprezentuje obiekty jednego typu (rodzaju)
- Przykładami klas pojęciowych są: *Film*, *Seans* oraz *Sala*
- Nazwy klas pojęciowych znajdują się w pierwszym polu prostokąta i pisze się je wielką literą
- Każdy człon nazwy zaczyna się wielką literą np. *RozkładFilmów*

Obiektowe modelowanie dziedziny

- **Atrybuty** klasy pojęciowej
- Atrybuty to napisy, liczby, daty bądź wartości logiczne opisujące poszczególne egzemplarze klasy pojęciowej. Każdy egzemplarz/obiekt klasy pojęciowej *Film* posiada na przykład *tytuł* i *czasTrwania*

Obiektowe modelowanie dziedziny

- **Powiązanie** oznacza, że między egzemplarzami klas pojęciowych może zachodzić związek
- Powiązania mają nazwy. Zapisuje się je tak samo jak nazwy klas pojęciowych, czasami nazwy członów rozdziela się myślnikami.
- Nazwy powiązań – zwroty czasownikowe
- Przykład:
  - film jest wyświetlany w trakcie seansu oraz
  - seans odbywa się w sali

Obiektowe modelowanie dziedziny

- Jak zacząć?
- Odnalezienie klas pojęciowych
- Analiza związków pomiędzy klasami pojęciowymi, i zapisanie ich jako powiązań
- Odnajdywanie atrybutów

Obiektowe modelowanie dziedziny

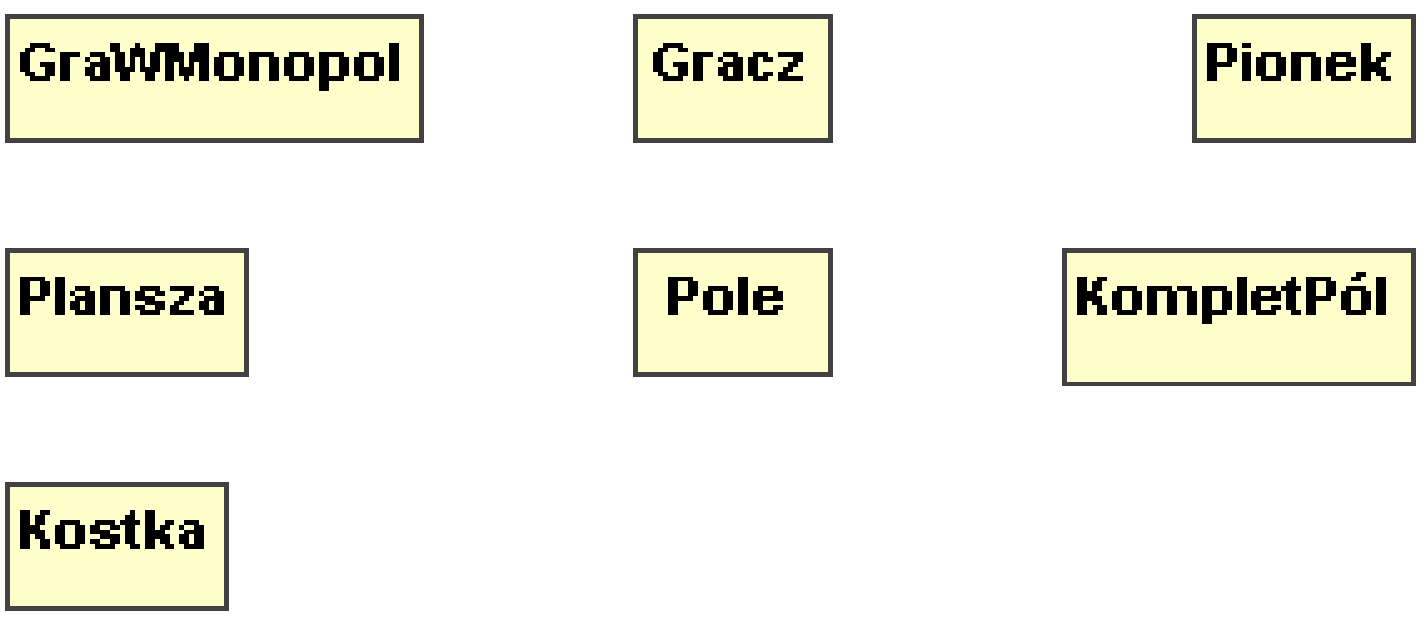
- Przy znajdowaniu **klas pojęciowych** (*ang. conceptual class*) należy:
  - używać istniejących nazw, np. gdy analizujesz system do zbierania wyników w nauce, który ma być używany w liceum, jego użytkownikami będą uczniowie, a gdy jest przeznaczony dla uczelni wyższej to studenci,
  - nie zajmować się niczym, co nie dotyczy modelowanej części rzeczywistości; jeżeli pracujesz iteracyjnie powstrzymaj się od rozpoznawania klas pojęciowych, które są nieistotne w obecnej iteracji oraz
  - nie dodawać rzeczy, których nie ma

Obiektowe modelowanie dziedziny

Kategoria	Przykłady	Klasy z dziedziny gry w Monopol
transakcje	*SprzedażBiletu*, *RezerwacjaMiejsc*	
pozycje transakcji Do tej kategorii jest dostępny komentarz [pokaż]	*PozycjaRezerwacji*	
produkty bądź usługi związane z transakcjami i kontraktami lub ich pozycjami	*Bilet*	
gdzie transakcje są odnotowywane	*Kasa*, *WykazDostępnychMiejsc*	
role ludzi i organizacji związanych z transakcją	*Kasjer*	*Gracz*
miejsce zajścia transakcji/obsługi transakcji	*Kasa*, *SalaKinowa*	
zdarzenia (często trzeba pamiętać czas ich zajścia)	*Seans*	*GraWMonopol*
obiekty fizyczne	*Bilet*, *Kino*, *Kasa*, *Miejsce*	*Plansza*, *Pionek*, *Kostka*
opisy Do tej kategorii jest dostępny komentarz [pokaż]	*OpisSeansu*	
katalogi	*KatalogSeansów*, *KatalogFilmów*	
kontenery rzeczy fizycznych lub informacji	*Kino*, *SalaKinowa*	*Plansza*
rzeczy w kontenerach	*SalaKinowa*, *Miejsce*	*Pole*
inne współpracujące systemy	*SystemAutoryzującyPłatnościElektroniczne*	
potwierdzenia, rejestry, kontrakty, zagadnienia prawne	*Pokwitowanie*, *PotwierdzenieRezerwacji*	
instrumenty finansowe	*Czek*, *Gotówka*	
harmonogramy, instrukcje, dokumenty regularnie używane podczas wykonywania prac	*DziennaListaPromocji*	

Obiektowe modelowanie dziedziny

- Jak definiować klasy pojęciowe?
  - analiza fraz rzeczownikowych w tekstowym opisie dziedziny lub wymagań



Obiektowe modelowanie dziedziny

- **Powiązanie** (ang. association) między klasami wskazuje, że między ich egzemplarzami może występować jakaś zależność
  - W modelu dziedziny pokazujemy powiązania, które są niezbędne do wypełnienia wymagań informacyjnych i pomagają zrozumieć dziedzinę.
  - Duża ilość powiązań - diagramy mało czytelne
  - Ważne są powiązania między klasami, jeżeli przez jakiś czas "trzeba pamiętać" o zależności między ich egzemplarzami
  - powiązania rysuje się między klasami ale chodzi o ich egzemplarze

Nazwy powiązań:

Należy pamiętać by nazywać powiązania zgodnie z ich faktycznym przeznaczeniem określając kontekst występowania. Określenie *„egzemplarze klasy A dotyczą egzemplarzy klasy B”*, są mało precyzyjne i nie określają w sposób jednoznaczny oraz zrozumiały znaczenia.

Obiektowe modelowanie dziedziny

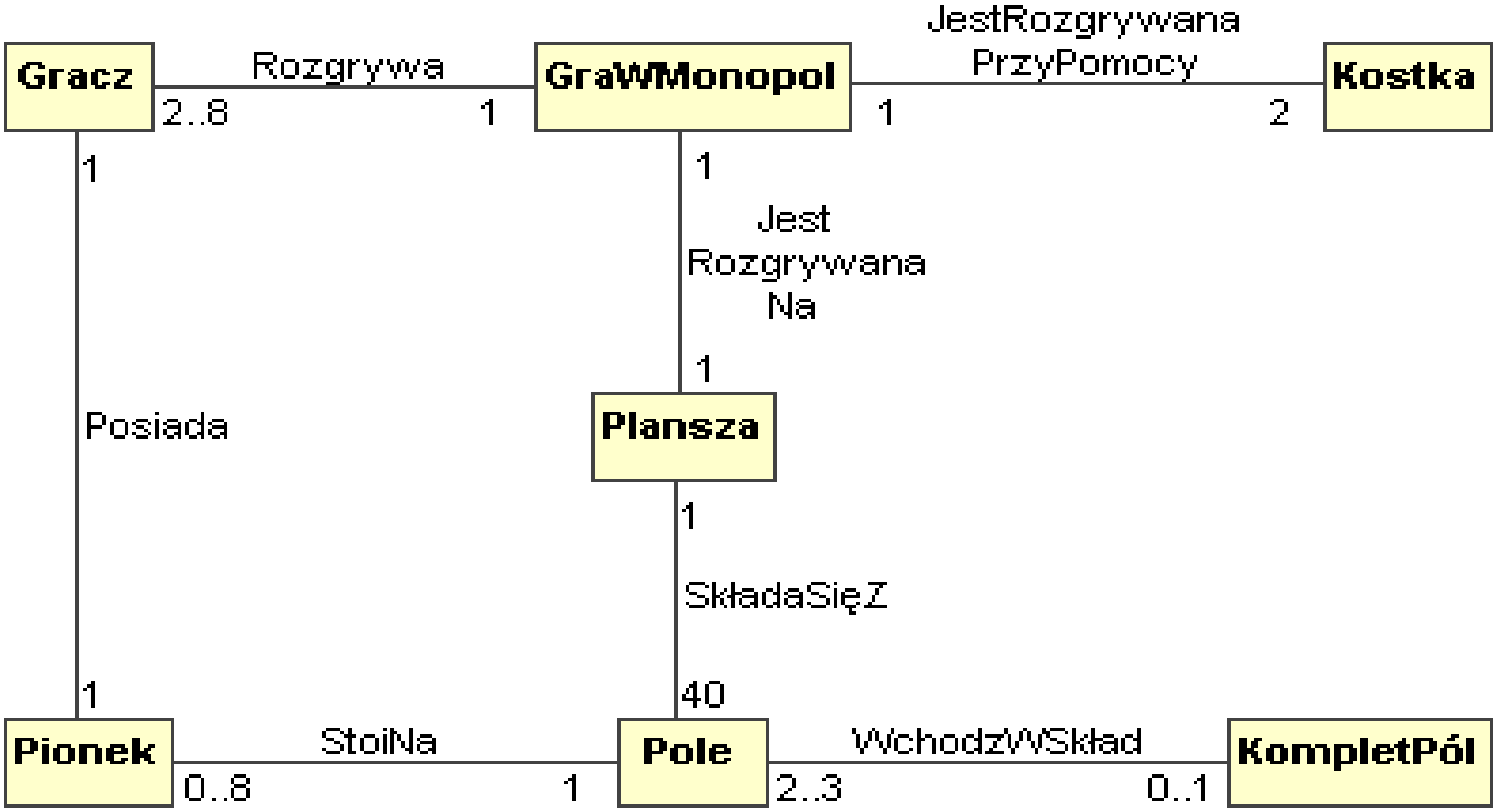
Kategoria	Przykłady	Klasy z dziedziny gry w Monopol
A jest transakcją związaną z inną transakcją B	*Płatność–RezerwacjaMiejsc*	
A jest pozycją transakcji B	*PozycjaRezerwacji–RezerwacjaMiejsc*	
A jest produktem lub usługą z transakcji lub pozycji transakcji B	*Bilet–SprzedażBiletu*	
A jest rolą związaną z transakcją B	*Klient–Płatność*	
A jest fizyczną lub logiczną częścią B	*Miejsce–SalaKinowa, SalaKinowa–Kino*	*Pole–Plansza, Pole–KompletPól, GraWMonopol–Plansza, GraWMonopol–Kostka, GraWMonopol–Pionek*
A fizycznie lub logicznie przechowywane w/na B	*Kasa–Kino*	*Pionek–Pole, Pole–Plansza*
A jest opisem B	*OpisSeansu–Seans*	
A jest rejestrowane, zgłaszane, utrwalane, pamiętane w/na B	*SprzedażBiletu–Kasa*	*Pionek–Pole, GraWMonopol–Plansza*
A jest uczestnikiem/pracownikiem/członkiem B	*Kasjer–Kino*	*Gracz–GraWMonopol*
A jest organizacyjną podjednostką B	*Kino–SiećKin*	
A używa, zarządza lub posiada B	*Kasjer–Kasa*	*Gracz–Pionek*
A jest obok B	*PozycjaRezerwacji–PozycjaRezerwacji*	

Obiektowe modelowanie dziedziny

- **Liczebność** (*ang. multiplicity*) określa: *„jak wiele egzemplarzy klasy A może być powiązane z jednym egzemplarzem klasy B”*. Na diagramach liczebność przedstawia się w postaci wyrażenia umieszczanego obok klasy A obok linii obrazującej powiązanie. Przykład:
  - 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)
  - * (dowolna ilość również zero)

Obiektowe modelowanie dziedziny

- **Przykład (**klasy pojęciowe oraz powiązania wraz z nazwami i liczebnościami**):**

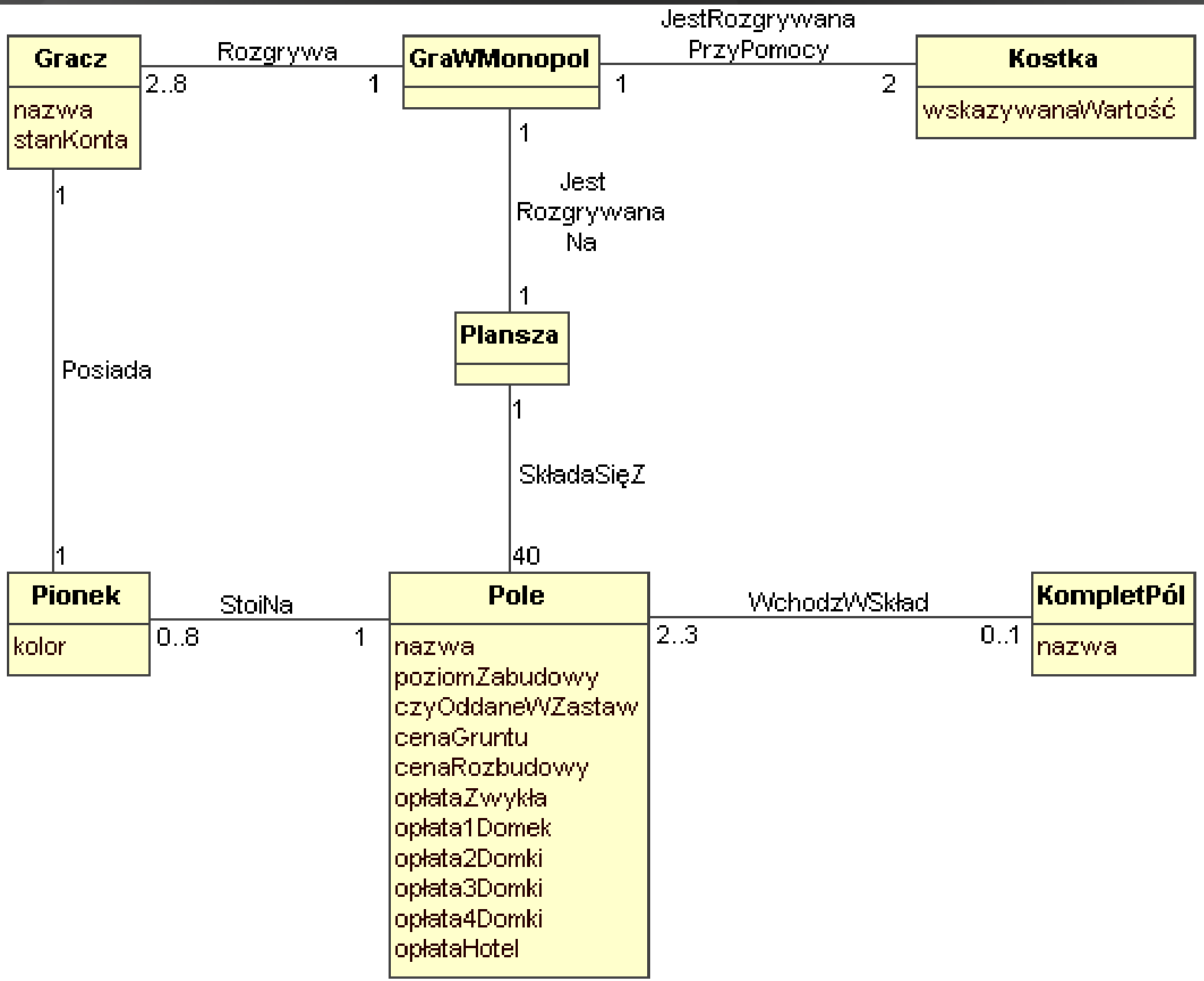


Obiektowe modelowanie dziedziny

- Dodawanie atrybutów do klas pojęciowych:
  - **Atrybuty** (*ang. attirbute*) opisują egzemplarze klas pojęciowych. Nie wszystkie klasy pojęciowe muszą mieć atrybuty.
  - Atrybuty - wartości typów podstawowych "czyste dane" np. napisy, liczby, wartości logiczne czy daty

    Przykład:
    Przykładowo *kierownik* nie jest atrybutem *Kina*. To oddzielna klasa pojęciowa sama posiadająca wiele atrybutów jak *imię, nazwisko* czy *dataUrodzenia*.

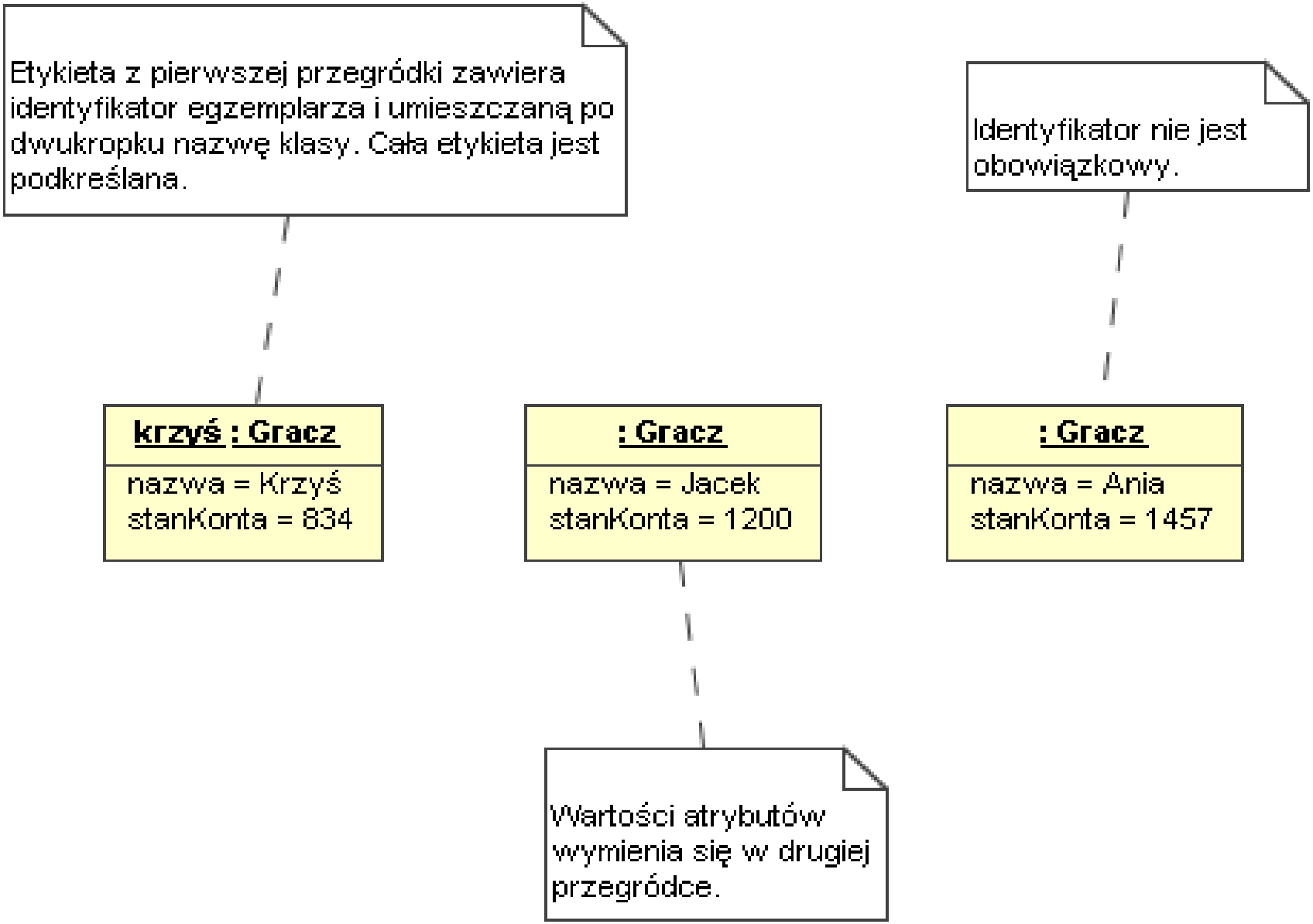
Obiektowe modelowanie dziedziny



Obiekty i ich stan – przykładowa rozgrywka

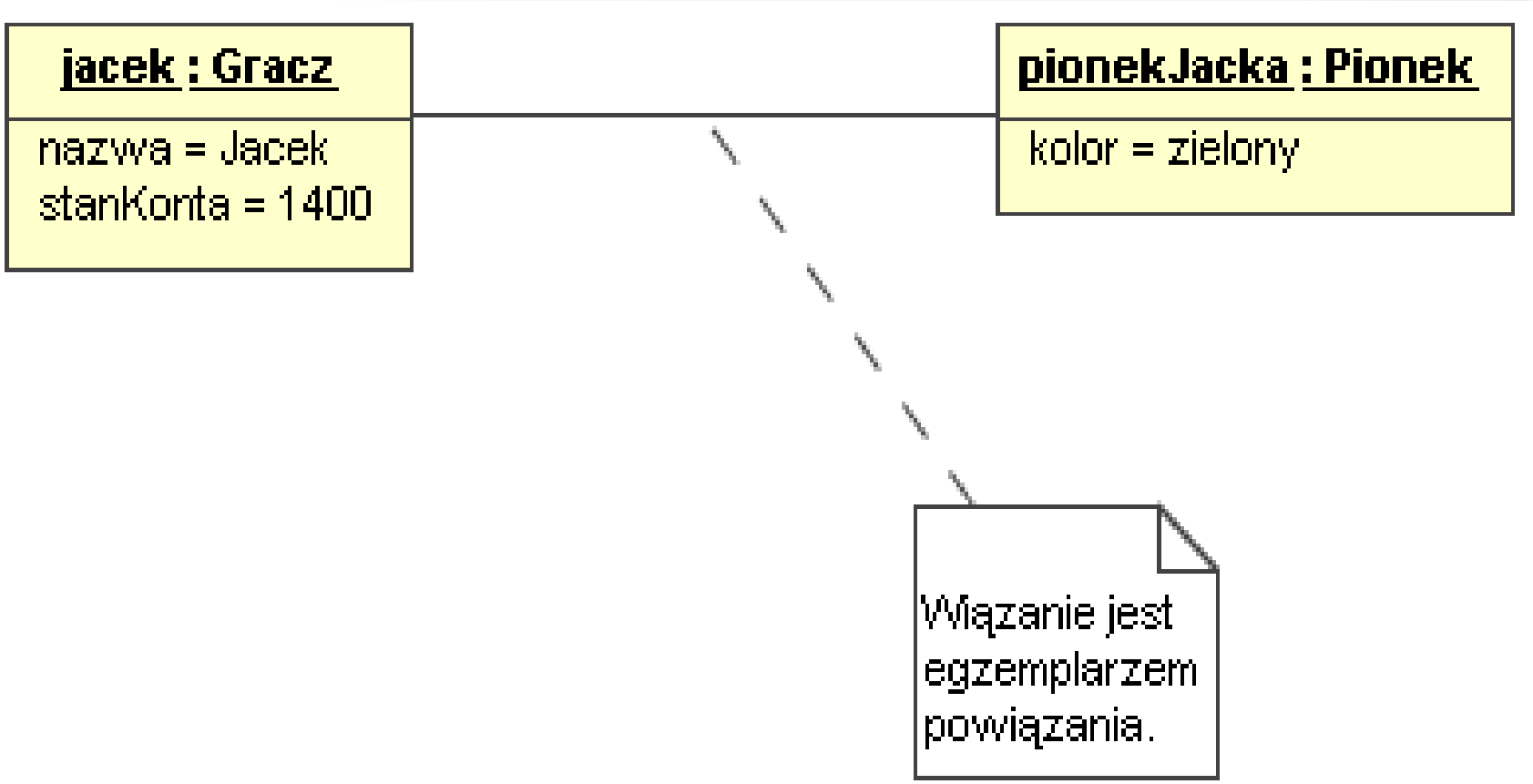
- Stan obiektu to wartości jego atrybutów
- Stany wszystkich obiektów to stan programu
- Przykład:
  - Pierwsze pole zawiera `[<id_obiektu>] : <nazwa_klasy>`
  - Drugie pole zawiera atrybuty wraz z ich wartościami `<nazwa_atrybutu> = <wartość>`
  - notatki

Obiekty i ich stan – przykładowa rozgrywka



Obiekty i ich stan – przykładowa rozgrywka

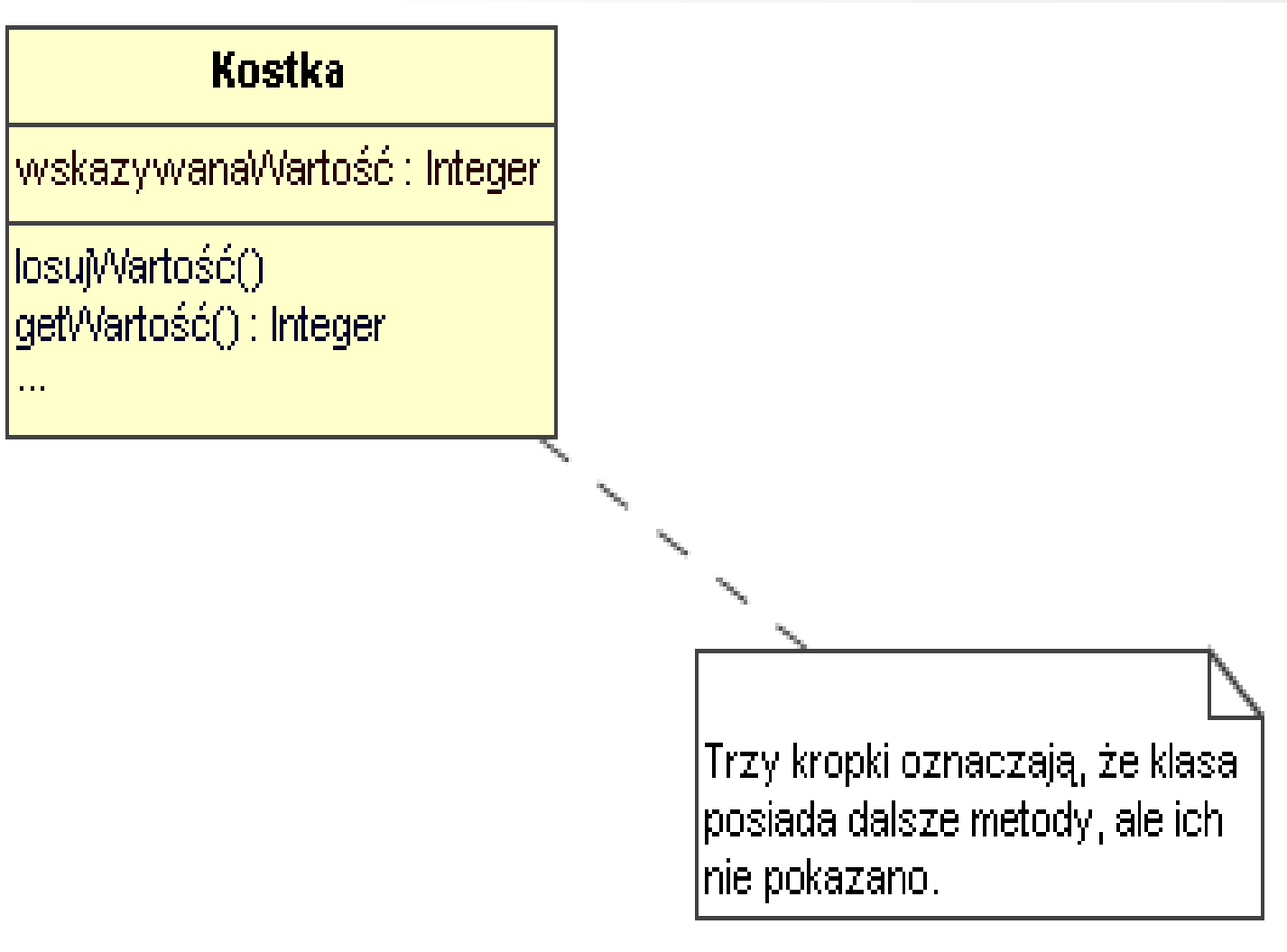
- Godne uwagi związki pomiędzy obiektami pokazujemy w ten sam sposób co powiązania między klasami. Przyjęło się je nazywać **wiązaniami** (*ang. link*).



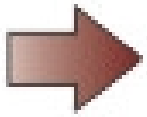
Obiekty w programowaniu

- Klasa = typ danych
- Obiekt = wartość określonego typu
- Organizacja kodu operującego na danych:
  - Dane i kod na nich operujący zebrane są razem (w klasie)
  - Kod w każdej klasie jest podzielony na metody, które wywołuje się na rzecz jakiegoś egzemplarza
  - Metody mogą mieć parametry i dawać jakieś wartości, mają bezpośredni dostęp do danych egzemplarza, na rzecz którego są wykonywane, oraz mogą te dane modyfikować
- Atrybuty i metody jednej klasy nazywa się jej **składowymi** (*ang. class member*).

Obiekty w programowaniu – to już nie jest modelowanie dziedziny



Obiekty w programowaniu – klasa pojęciowa → klasa projektowa



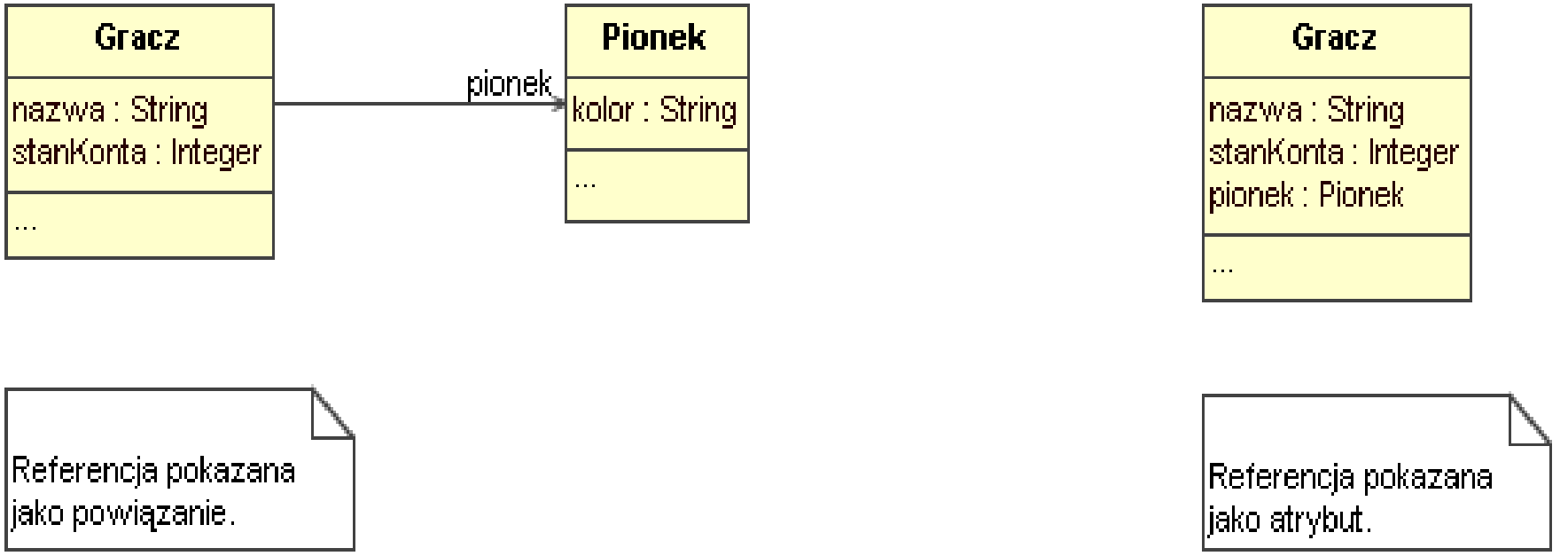
Kostka
-wskazywanaWartość : Integer
+losujWartość() +getWartość() : Integer ...

Obiekty w programowaniu – to już nie jest modelowanie dziedziny

- **klasa projektowa** (*ang. design class*), które przedstawiają projekt oprogramowania
- programowanie obiektowe polega na wyznaczaniu obiektom odpowiedzialności:
  - za pamiętanie pewnych danych
  - za wykonywanie operacji na tych danych
- Wszystkie szczegóły, w jaki sposób obiekty wywiązują się z tych odpowiedzialności, są ukryte w definicji klasy
- Pisząc metody danej klasy, możemy się skupić tylko na wycinku całego zadania – na realizacji odpowiedzialności wyznaczonych tej klasie
- operacje na danych innych obiektów = zlecenie ich

Jak zapisać powiązania?

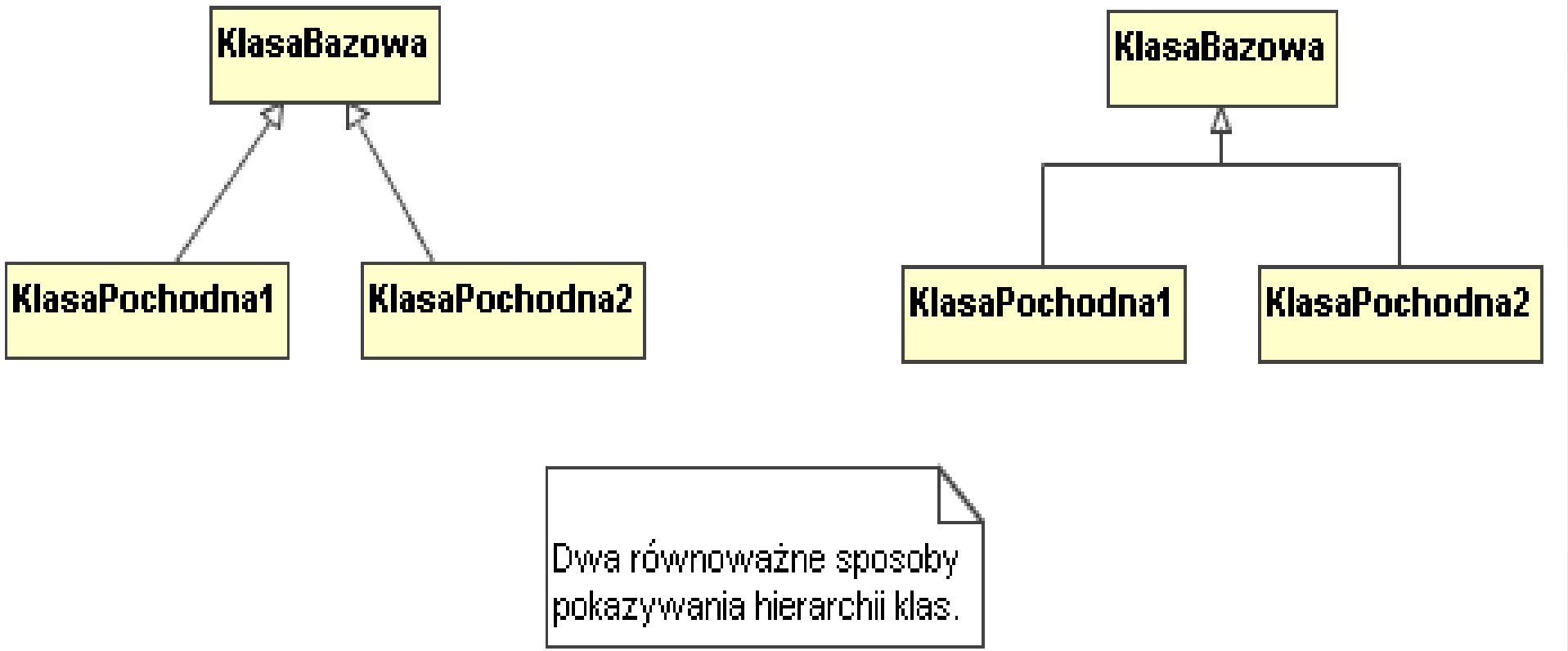
- jedna z klas uczestniczących w powiązaniu będzie posiadała atrybut, na który zostanie przypisany obiekt drugiej klasy
- atrybut ten nazywa się **referencją** (*ang. reference*)



Hierarchia klas

- potrzeba kilku wariantów pewnej klasy
- Korzystanie bez zmian
- Inne działanie
- **Klasa bazowa** (*ang. base class*) bądź inaczej **nadklasa** (*ang. superclass*) definiuje składowe wspólne dla wszystkich wariantów
- **Klasy pochodne** (*ang. derived class*) bądź inaczej **podklasy** (*ang. subclass*) definiują pozostałe składowe, które występują tylko w poszczególnych wariantach
- związek między nadklasą i podklasami nazywany jest **uogólnieniem** (*ang. generalization*), bądź związkiem **uogólnienie-uszczegółowienie**

Hierarchia klas

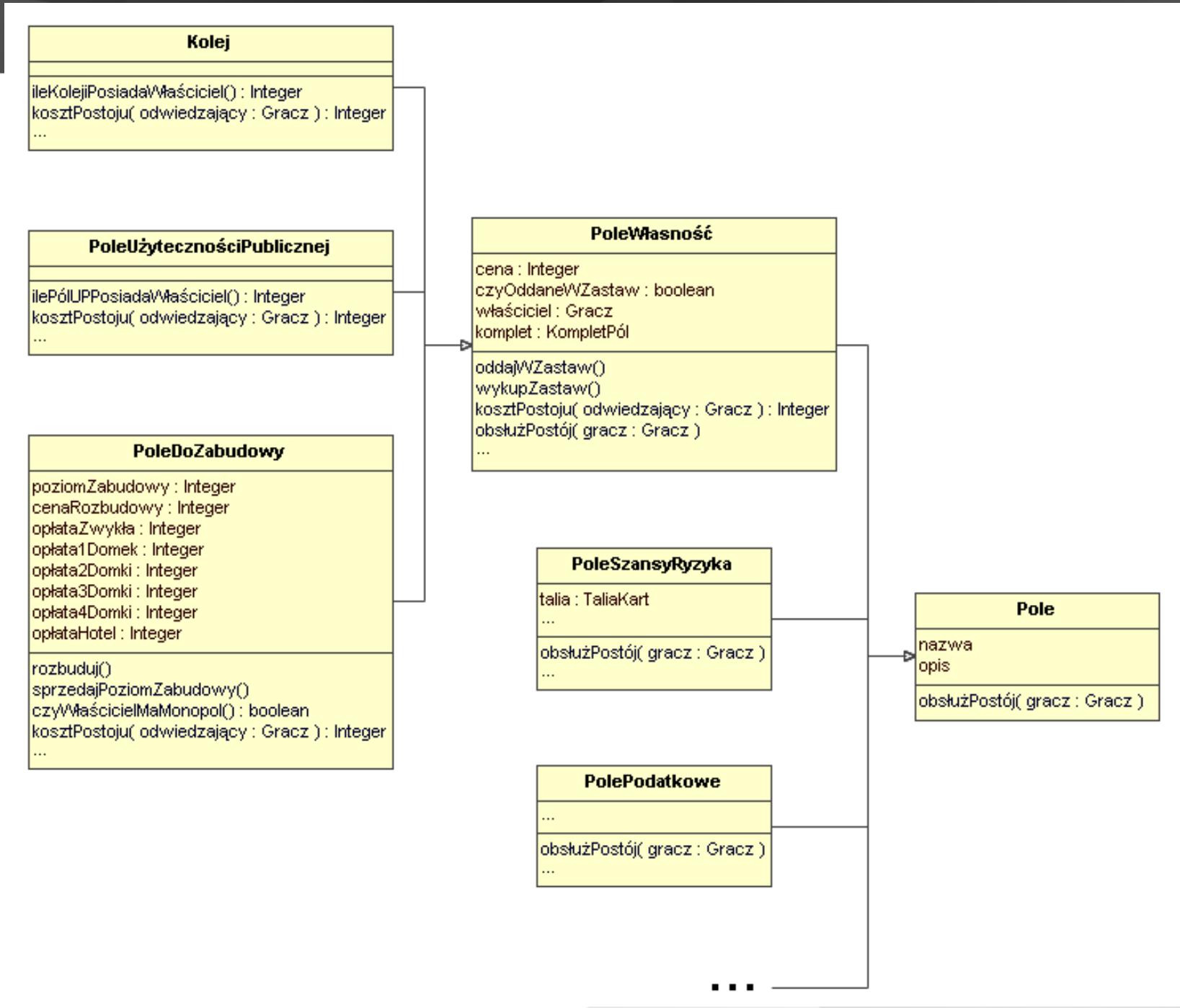


- obiekty podklasy są również egzemplarzami nadklasy i powinny dawać się używać w tych samych kontekstach

Dziedziczenie

- klasa pochodna **dziedziczy** (*ang. inherits*) wszystkie składowe klasy bazowej
- klasa pochodna może zachowanie niektórych metod **przedefiniować** (*ang. override*) podając nowe definicje
- podklasa **rozszerza** (*ang. extend*) nadklasę gdyż może zawierać nowe składowe
- na diagramie jak klasa pochodna ponownie definiuje metodę klasy bazowej należy to zapisać umieszczając jej definicję w klasie pochodnej

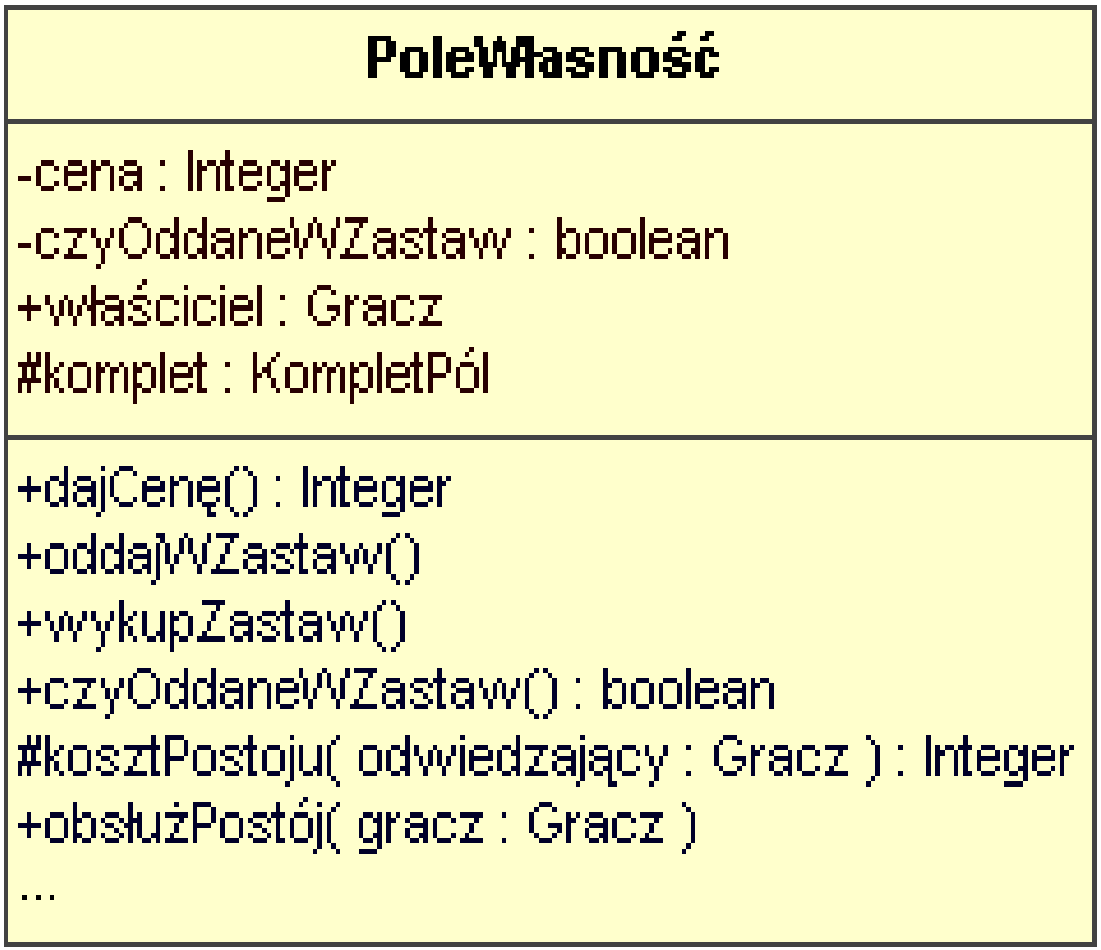
Dziedziczenie



Kontrakty i widoczność

- Kontrakt - odpowiedzialność za wykonanie danej funkcjonalności
- Urywanie metod oraz atrybutów przed innymi obiektami - zakresy widoczności
- W celu separacji szczegółów kontraktu od szczegółów implementacji w językach obiektowych występuje pojęcie **widoczności** (*ang. visibility*):
  - **publiczne** (*ang. public*) – mogą ich bez ograniczeń używać obiekty wszystkich klas „+”
  - **chronione** (*ang. protected*) – mogą ich bez ograniczeń używać obiekty tej samej klasy lub jej podklas „#”
  - **prywatne** (*ang. private*) – mogą ich używać jedynie obiekty tej samej klasy „-”

Kontrakty i widoczność



Kapsułkowanie

- Przed rozpowszechnieniem programowania obiektowego, wszystkie dane były globalnie dostępne i można było na nich operować w dowolny sposób i z każdego miejsca w kodzie → błędy
- Niespójny stan zmiennych
- Ukrywanie danych przez obiekty nazywa się kapsułkowaniem
- Ograniczenie - wykonywanie na danych operacji tylko określonego rodzaju (np. odczyt)
- zmiany w implementacji, np. struktury danych → ograniczony zasięg

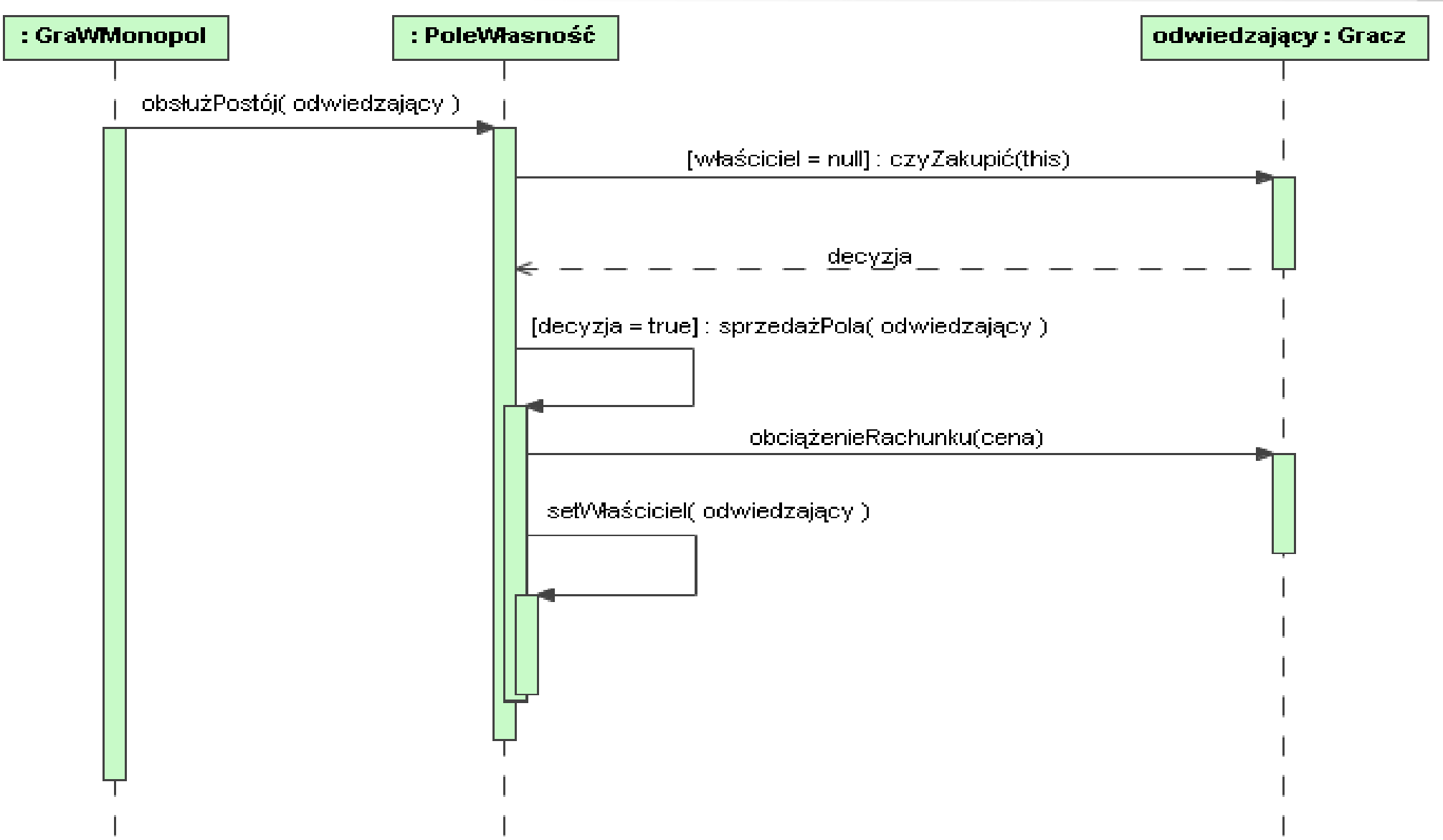
Kapsułkowanie - konwencja

- W wielu językach programowania obiektowego (m.in. JAVA) używa się pewnej konwencji:
  - `nazwaPola`
  - `getNazwaPola` – pobiera pole
  - `setNazwaPola` – ustawia wartość pola
  - `isNazwaPola` – dla wartości logicznej
- Język C#:
  - Wykorzystywane są własności:
    - `set {}`
    - `get {}`

Diagramy przebiegu

- **diagramy klas** – określenie kontraktu, czyli odpowiedzialności klas
- Interakcja pomiędzy obiektami – opis jak mają ze sobą współpracować – **diagram przebiegu**
- Diagram przebiegu określa:
  - Kto zleca czynność
  - Kto wykonuje czynność
  - O jaką czynność chodzi

Diagramy przebiegu



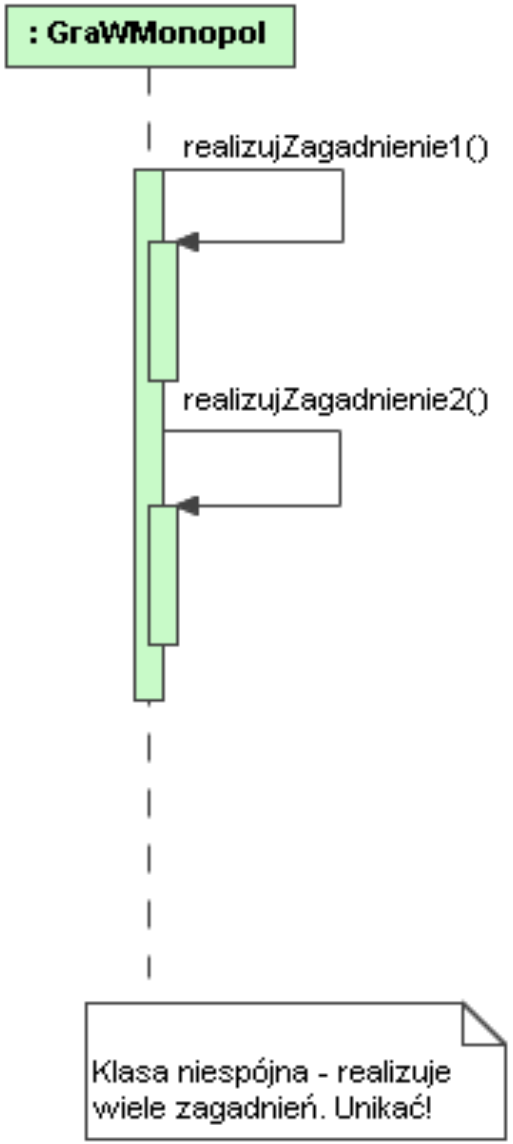
Sprzężenie

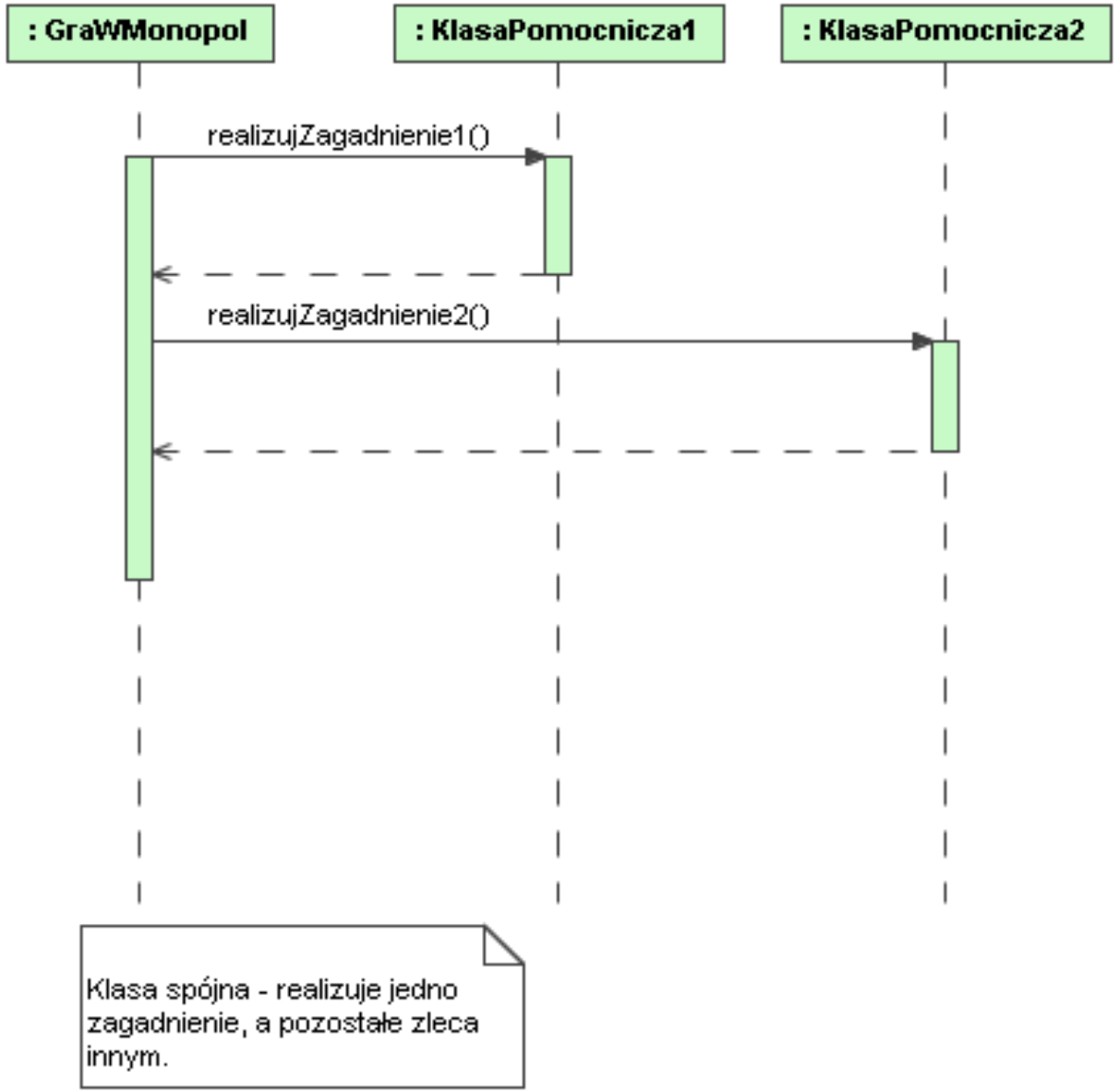
- **Sprzężenie** (*ang. coupling*) jest miarą jak bardzo obiekty, podsystemy lub systemy zależą od siebie nawzajem
- obiekt wykonujący metodę innego obiektu jest z nim sprzężony
- podklasa jest sprzężona z nadklasą
- Sprzężenie obiektów utrudnia wprowadzanie zmian w kodzie
- Im luźniej obiekty są ze sobą sprzężone, tym zasięg potencjalnych problemów wywołanych zmianą jednego z nich jest mniejszy
- Rada: minimalizacja sprzężenia

Spójność

- **Spójność** (*ang. cohesion*) to miara jak funkcjonalnie powiązane są metody danej klasy
- Warto dbać o utrzymanie **wysokiej spójności** (*ang. high cohesion*)
- Jeśli jedna klasa odpowiada za zbyt wiele zagadnień (jest niespójna), z czasem nadmiernie się rozrośnie i będzie trudna do zrozumienia oraz pielęgnacji
- Należy tworzyć klasy pomocnicze, którym będzie zlecana część odpowiedzialności

Spójność - przykład





Wzorce projektowe

- Przy wyznaczaniu odpowiedzialności obiektom istnieje wiele poziomów swobody
- Każde rozwiązanie ma silne i słabe strony
- Niektóre są łatwiejsze w pielęgnacji, a inne bardziej elastyczne
- sprawdzone pomysły o dobrze poznanych silnych i słabych stronach nazywa się **wzorcami projektowymi** (*ang. design pattern*)
- Wzorce projektowe posiadają nazwy jak Fabryka (*ang. Factory*), Singleton (*ang. Singleton*), Strategia (*ang. Strategy*) i Fasada (*ang. Facade*)
- Dobrze dobrane i ogólnie przyjęte nazwy wzorców ułatwiają ich zapamiętywanie oraz usprawniają komunikację międzyludzką (np. w zespole programistycznym dyskutującym jak rozwiązać nowy problem)

UML

- Notacja graficzna, przedstawiona na wykładzie to Unified Modeling Language (UML)
- jest to graficzny język opracowany specjalnie w celu specyfikowania i planowania artefaktów programistycznych oraz usprawnienia komunikacji między członkami zespołu programistycznego
- Prace nad UML zapoczątkowali - Grady Booch, Jim Rambough i Ivar Jacobson
- UML jest powszechnie stosowany, a od 1997 roku jest standardem Object Management Group (OMG) i doczekał się już drugiej wersji
- Największą zaletą UML jest to, iż jest to notacja graficzna
- Rozpoznawanie symboli jest silną stroną ludzkiego mózgu, dlatego oglądając prostokąty i linie na diagramach łatwo zrozumieć zależności między obiektami

UML

- UML pozwala skupić uwagę na najważniejszych koncepcjach lub pomysłach i pominąć bądź ukryć nieistotne szczegóły
- UML to tylko notacja
- UML jest bardzo rozbudowany, zawiera kilkanaście rodzajów diagramów

UML - wersje

Wersje opublikowane przez RSC i UML Partners

0.9	1995
1.0	Styczeń 1997

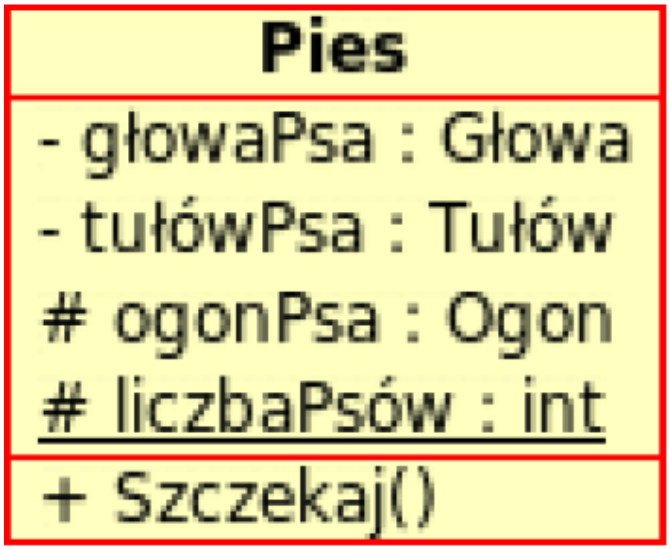
Wersje UML opublikowane przez OMG

Wersja	Data publikacji	URL
1.3	Marzec 2000	http://www.omg.org/spec/UML/1.3
1.4	Wrzesień 2001	http://www.omg.org/spec/UML/1.4
1.4.2	Lipiec 2004	zobacz: ISO/IEC 19501
1.5	Marzec 2003	http://www.omg.org/spec/UML/1.5
2.0	Lipiec 2005	http://www.omg.org/spec/UML/2.0
2.1.1	Sierpień 2007	http://www.omg.org/spec/UML/2.1.1
2.1.2	Listopad 2007	http://www.omg.org/spec/UML/2.1.2
2.2	Luty 2009	http://www.omg.org/spec/UML/2.2/

Wersje UML opublikowane przez ISO

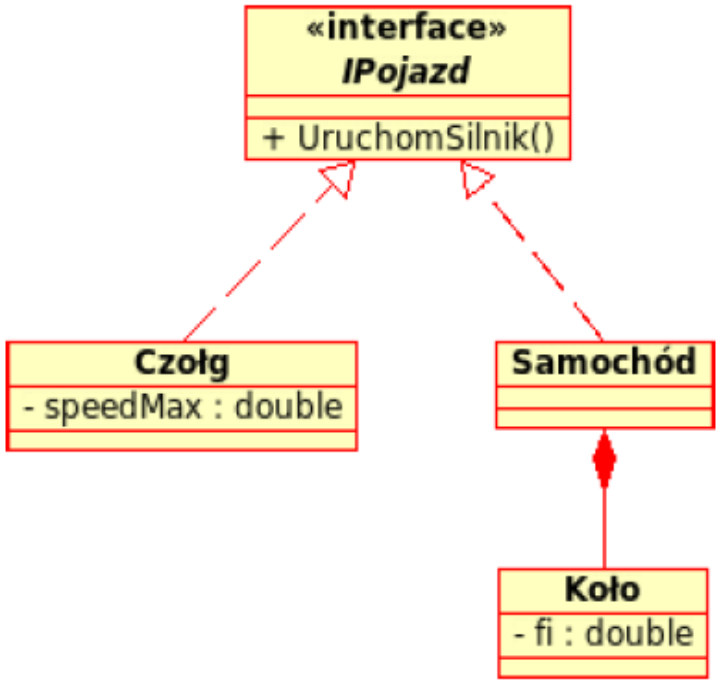
Numer	Data	Wersja	URL
ISO/IEC 19501	Styczeń 2005	1.4.2	http://www.omg.org/spec/UML/ISO/19501/PDF

UML → kod




```
public class Pies {
    private Głowa głowaPsa;
    private Tułów tułówPsa;
    protected Ogon ogonPsa;
    protected static liczbaPsów;

    public void Szczekaj ()
      { ... };
}
```

UML → kod cd.

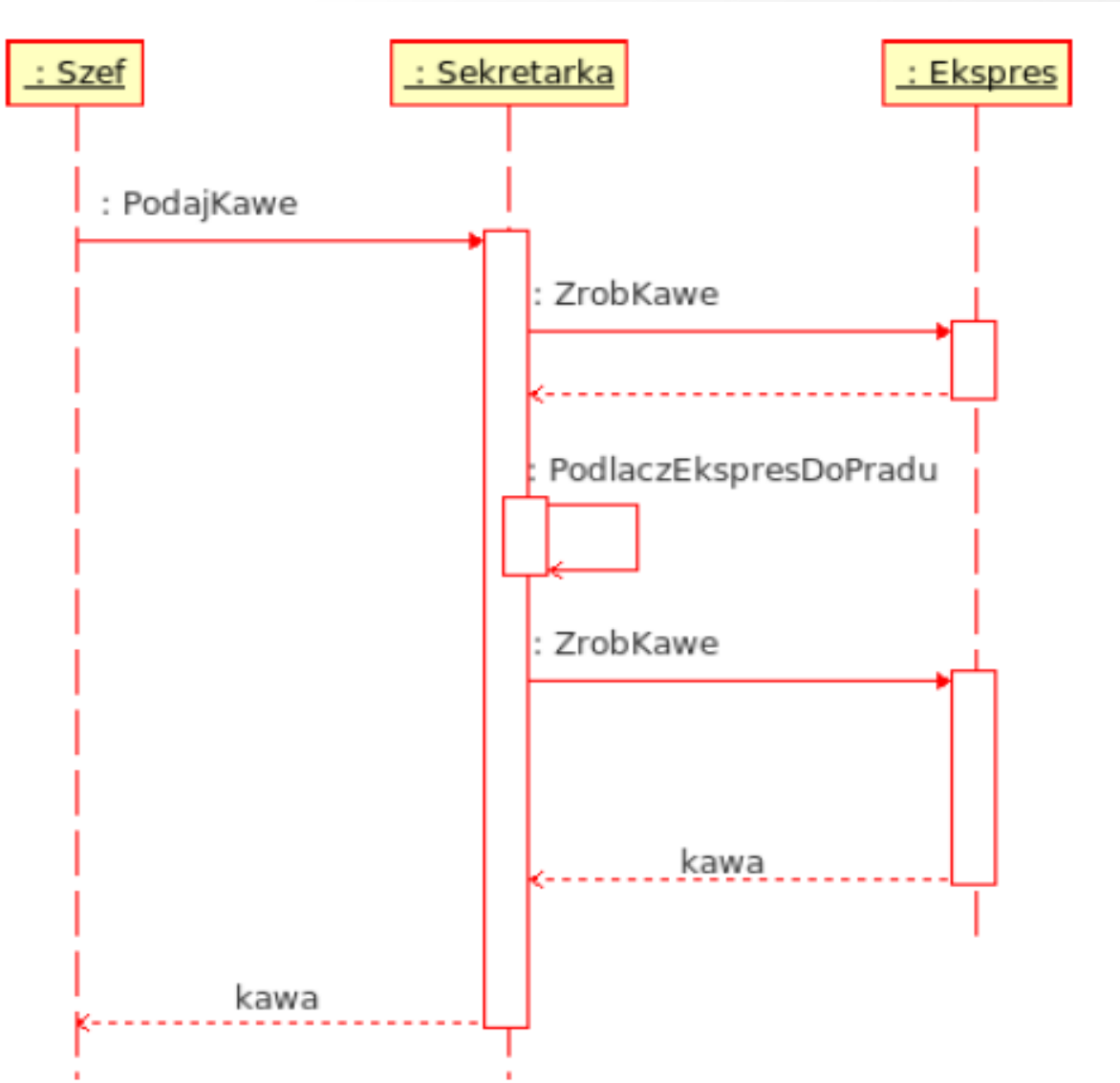



```
public interface IPojazd {
    public void UruchomSilnik ();
}

public class Samochód : IPojazd {
    private Koło[ ] koła;
    public override void UruchomSilnik
      () {...}
}

public class Czołg : IPojazd {
    private double speedMax;
    public override void UruchomSilnik
      () {...}
}
```

Diagramy przebiegu

- „Szef prosi sekretarkę o kawę”
  - obiekt *Szef* wywołuje metodę *PodajKawe* obiektu *Sekretarka*,
  - *Sekretarka* wywołuje metodę *ZróbKawe* obiektu Ekspres
  - *Sekretarka* wywołuje własną metodę PodłączEkspresDoPrądu
  - *Sekretarka* wywołuje ponownie metodę *ZróbKawe* obiektu *Ekspres*
  - *Sekretarka odbiera obiekt Kawa od Ekspres i przekazuje do Szef*

Diagramy przebiegu → UML



Karty CRC

- **C**lass **R**esponsibilities and **C**ollaborators
- wprowadzone przez Kenta Becka i Warda Cunnighama i opisane w artykule *„A Laboratory for Teaching Object-Oriented Thinking”* przedstawionym na konferencji OOPSLA’89
- Narzędzie wykorzystywane podczas burzy mózgów do identyfikowania klas
- modeluje się behawioralne własności budowanego systemu informatycznego

Karty CRC

Class name	
Superclasses	
Subclasses	
Responsibilities	Collaborators
Author	

Karty CRC

Klient	
Składa zamówienia	Zamówienie
Opłaca faktury	Faktura
Zna swoją nazwę	
Zna swój adres	

Zamówienie	
Zna numer	Klient
Zna datę złożenia	Faktura
Zna wartość	Pozycja
Zna pozycje	
Zna klienta	