Kurs dokształcający
Python dla początkujących

Programowanie obiektowe

Wprowadzenie

dr inż. Michał Bednarczyk

Wydział Geoinżynierii, UWM w Olsztynie
ul. Heweliusza 12 pok. 26 (II piętro)
email: michal.bednarczyk@uwm.edu.pl

Cel kształcenia i treści merytoryczne

Celem zajęć jest zapoznanie uczestników kursu z zaawansowanymi właściwościami oraz możliwościami języka Python.

Treści merytoryczne:

- pojęcie klasy, obiektu,
- tworzenie obiektów, pojęcie konstruktora,
- metoda, różnica pomiędzy metodą a funkcją,
- hermetyzacja,
- dziedziczenie oraz polimorfizm,
- debugowanie i testowanie kodu.

Wprowadzenie

- Programowanie obiektowe (ang. object-oriented programming) jest obecnie najpopularniejszą techniką tworzenia programów komputerowych.
- Program komputerowy wyraża się jako zbiór obiektów będących bytami łączącymi stan (czyli dane) i zachowanie (czyli metody, które są procedurami operującymi na danych obiektu).
- W celu realizacji zadania obiekty wywołują nawzajem swoje metody, zlecając w ten sposób innym obiektom odpowiedzialność za pewne czynności.
- W porównaniu z tradycyjnym programowaniem proceduralnym, w którym dane i procedury nie są ze sobą powiązane, programowanie obiektowe ułatwia tworzenie dużych systemów, współpracę wielu programistów i ponowne wykorzystywanie istniejącego kodu.

Obiektowe modelowanie dziedziny

Historia

- Pierwszy język obiektowy – Simula 67 – powstał już w latach sześćdziesiątych XXw.
- Służył do symulacji statków. Dla każdego statku należało uwzględnić wiele atrybutów. Ponieważ liczba modelowanych rodzajów statków była duża, uwzględnienie wszystkich możliwych zależności między atrybutami stało się problematyczne. Pojawił się pomysł, aby pogrupować różne rodzaje statków w klasy obiektów.
- Każda klasa obiektów sama miała być odpowiedzialna za definiowanie swoich danych i zachowania.
- Simula była pierwszym językiem programowania, w którym wprowadzono pojęcie klasy i jej egzemplarza.
- Zgodnie z nazwą języka takie odwzorowanie obiektów spotykanych w świecie rzeczywistym na obiekty programowe można nazwać symulacją.

Historia

- Niedługo potem w laboratorium badawczym Xerox's Palo Alto stworzono Smalltalk.
- Smalltalk zawierał wiele rewolucyjnych pomysłów - m.in. dziedziczenie - i zyskał sobie sporą popularność.
- Jednak programowanie obiektowe stało się standardem przemysłowym dopiero w latach dziewięćdziesiątych za sprawą języka C++, który jest obiektowym rozszerzeniem C.
- Obecnie wiele języków programowania opartych jest na paradygmacie OOP lub stosuje jego elementy. Są to np. Java, Python, Ruby, Object Pascal, JavaScript, C#, PHP, R, Visual Basic …..

Co to znaczy „myśleć obiektowo”?

Czemu programowanie obiektowe jest tak popularne?

- Największym atutem programowania obiektowego jest zbliżenie programów komputerowych do naszego sposobu postrzegania rzeczywistości. Często nazywa się to zmniejszeniem luki reprezentacji (ang. representational gap).
- Wymyślając nowy lub analizując istniejący program obiektowy nasz mózg ma ułatwione zadanie. Dlatego ludzie są w stanie **łatwiej zapanować nad kodem** i tym samym **tworzyć większe programy**. Łatwiej jest również zrozumieć kod i pomysły innych programistów i tym samym **współpracować w zespole oraz ponownie wykorzystywać istniejące rozwiązania**.
- Klasyfikacja, czyli łączenie występujących w rzeczywistości obiektów w grupy – klasy, jest najbardziej naturalnym sposobem rozumienia rzeczywistości.

Analiza i projektowanie obiektowe

- Z programowaniem obiektowym nieodzownie wiążą się analiza i projektowanie obiektowe (ang. Object-Oriented Analysis and Design, OOA/D).
- **Analiza** to badanie problemu i wymagań, ale nie rozwiązania. Jest to termin o szerokim znaczeniu. 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. W wypadku systemu informacyjnego dla Zakładu Transportu Miejskiego obiektami pojęciowymi są na przykład: Autobus, Linia i Kierowca.
- **Projektowanie** to wymyślanie koncepcyjnego rozwiązania (programistycznego i sprzętowego), które realizuje wymagania. Takim koncepcyjnym rozwiązaniem może być opis schematu bazy danych lub klas programowych. Podczas projektowania zazwyczaj pomija się niskopoziomowe lub oczywiste (dla zamierzonych odbiorców projektu) szczegóły i koncentruje się na wysokopoziomowych pomysłach oraz ideach. Projektowanie obiektów programowych określane jest jako projektowanie obiektowe (ang. object-oriented design).

Analiza i projektowanie obiektowe

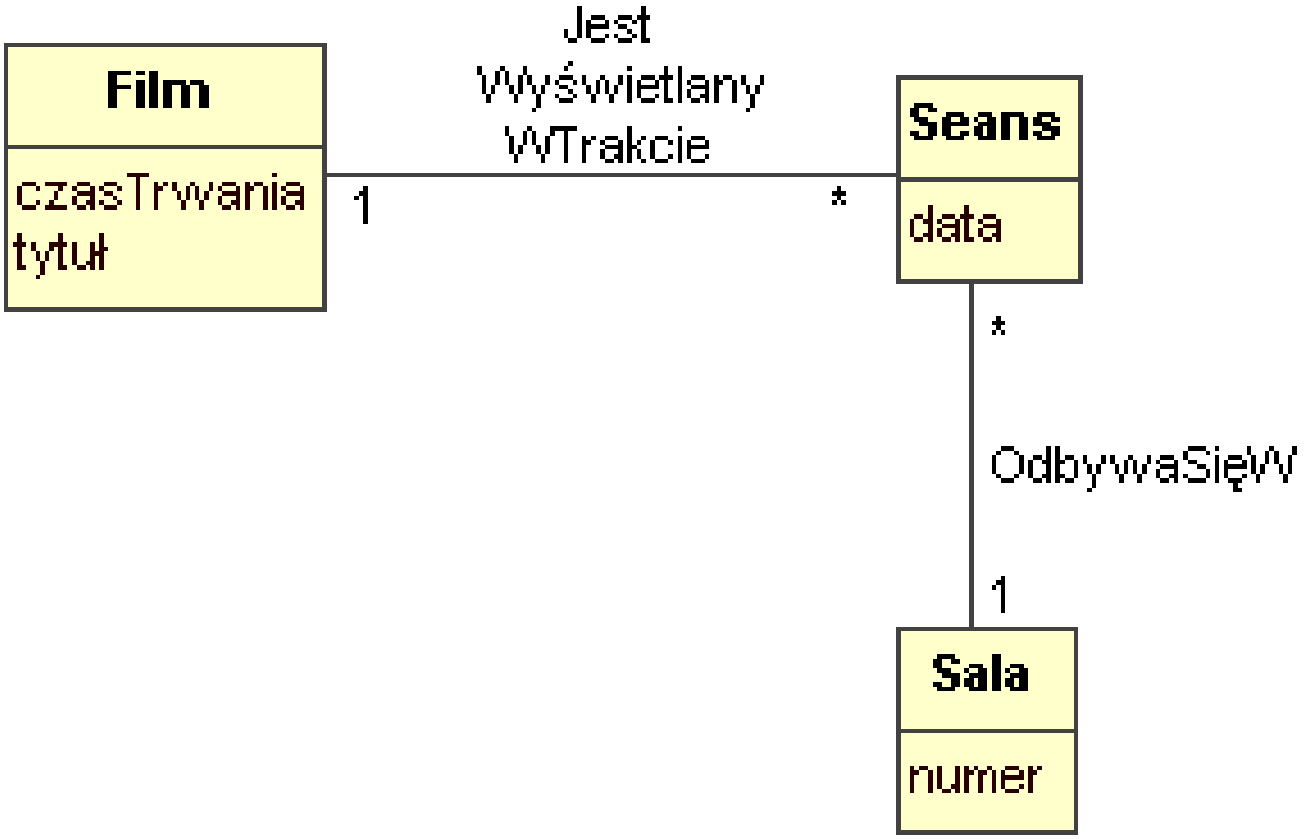
- Najważniejszą czynnością projektowania obiektowego jest **wyznaczenie odpowiedzialności obiektom programowym oraz określenie jak mają współpracować**, by wykonać zadanie.
- Analiza i projektowanie dają się krótko streścić jako "**zrób co należy** (analiza) oraz **zrób to jak należy** (projektowanie)".

Obiektowe modelowanie dziedziny

- Zgodnie ze swoją nazwą model dziedziny ma odzwierciedlać pojęcia z modelowanej części świata rzeczywistego oraz ich zależności. Model najczęściej wykonuje się w notacji **UML**.
- W modelu dziedziny nie zajmujemy się klasami programowymi (ang. software class). Może on posłużyć jako **źródło inspiracji** przy ich projektowaniu, ale nie w drugą stronę.
- Tworzenie modelu dziedziny nie jest obowiązkowe ale bardzo przydatne, jeżeli dziedzina problemu, który rozwiązujemy, nie jest dobrze zrozumiała.
- W modelu dziedziny pokazuje się:
  - klasy pojęciowe,
  - powiązania między klasami pojęciowymi i
  - atrybuty klas pojęciowych.

Przykład

Przykład częściowego modelu dziedziny dla aplikacji wspomagającej rezerwację miejsc i sprzedaż biletów kinowych.



Odnajdowanie klas pojęciowych

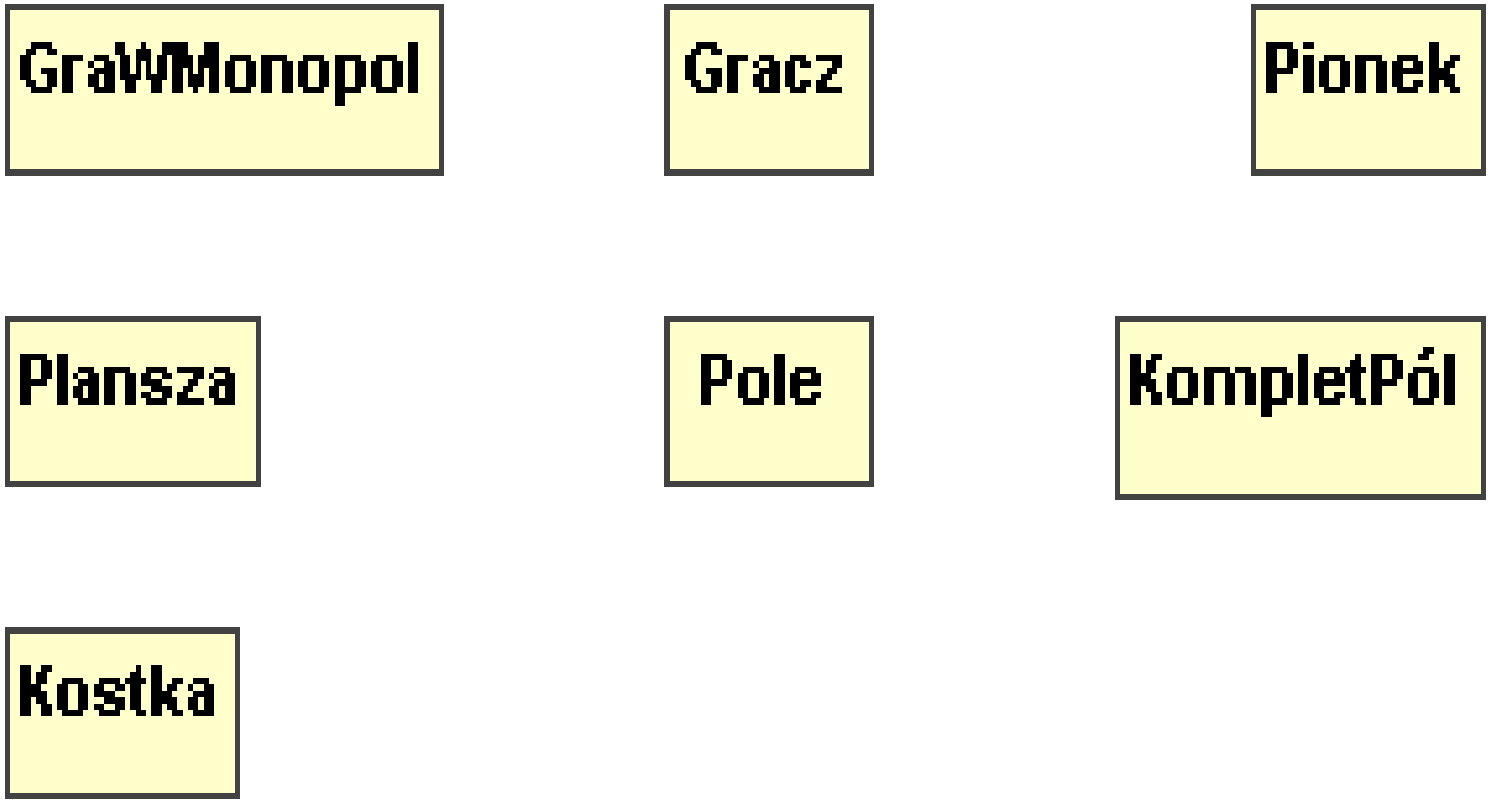
Przy znajdowaniu klas pojęciowych (ang. conceptual class) należy postępować jak kartograf:

- 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.**

Analiza fraz rzeczownikowych

- Wygodnym pomysłem na odnajdywanie klas pojęciowych jest analiza fraz rzeczownikowych w tekstowym opisie dziedziny lub wymagań (jeżeli takimi dysponujemy).
- W przypadku gry w Monopol na pewno warto rozpoznać frazy rzeczownikowe w instrukcji z zasadami gry.
- Do tego należałoby przeanalizować planszę i zawartość pudełka, nazywając poszczególne elementy i określając ich przeznaczenie (np. komplety pól, bank, karty szansy/ryzyka).

Wybrane klasy pojęciowe dla gry w Monopol



Modelowanie artefaktów programowych

- Pamiętajmy, że wykonujemy analizę dziedziny, a nie projekt.
- Nie uwzględniamy artefaktów programowych, np. okienek, baz danych, itp.
- Wyobraźmy sobie, że programu nigdy nie napiszemy lub że zrobi to ktoś inny.
- Wyjątkiem są artefakty programowe należące do dziedziny, którą modelujemy, na przykład można uwzględnić zewnętrzne systemy, z którymi nasz system będzie współpracował.

Odnajdowanie powiązań

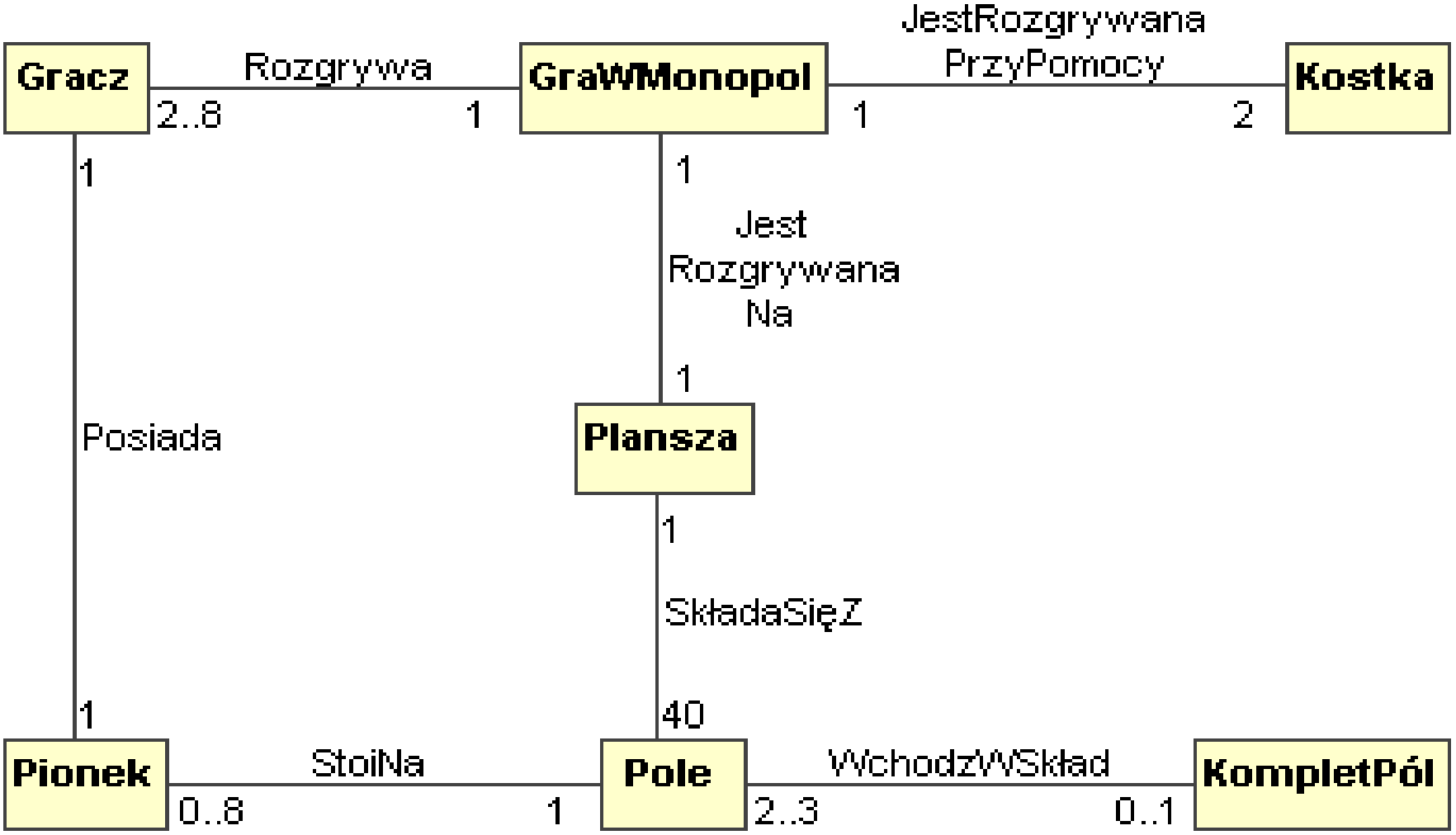
- Powiązanie (ang. association) między klasami wskazuje, że między ich egzemplarzami może występować jakaś zależność.
- W modelu dziedziny pokazujemy **powiązania, które są niezbędne do wypełnienia wymagań informacyjnych** i pomagają zrozumieć dziedzinę.
- Pokazanie **zbyt wielu powiązań** sprawi, że diagramy będą mało czytelne.
- Zazwyczaj warto pokazywać powiązania między klasami, jeżeli przez jakiś czas "trzeba pamiętać" o zależności między ich egzemplarzami. Dlatego, mimo że powiązania rysujemy między klasami, myślimy o ich egzemplarzach.
- Czy trzeba na przykład pamiętać na jakim Polu jest Pionek albo do jakiego Gracza należy? Oczywiście.
- Jednak informacji, że wartość wskazywana przez Kostkę określa, na które Pole się ruszyć, nie trzeba przechowywać w modelu. Jest to metainformacja. Tak samo nie ma potrzeby zapamiętywać, że Gracz przesuwa Pionek.

Liczebność

Odnajdując powiązania, powinniśmy zastanowić się nad ich liczebnością (ang. multiplicity). Liczebność 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 tuż przy linii obrazującej powiązanie. Przykładowe wyrażenia liczebności to:

- 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)

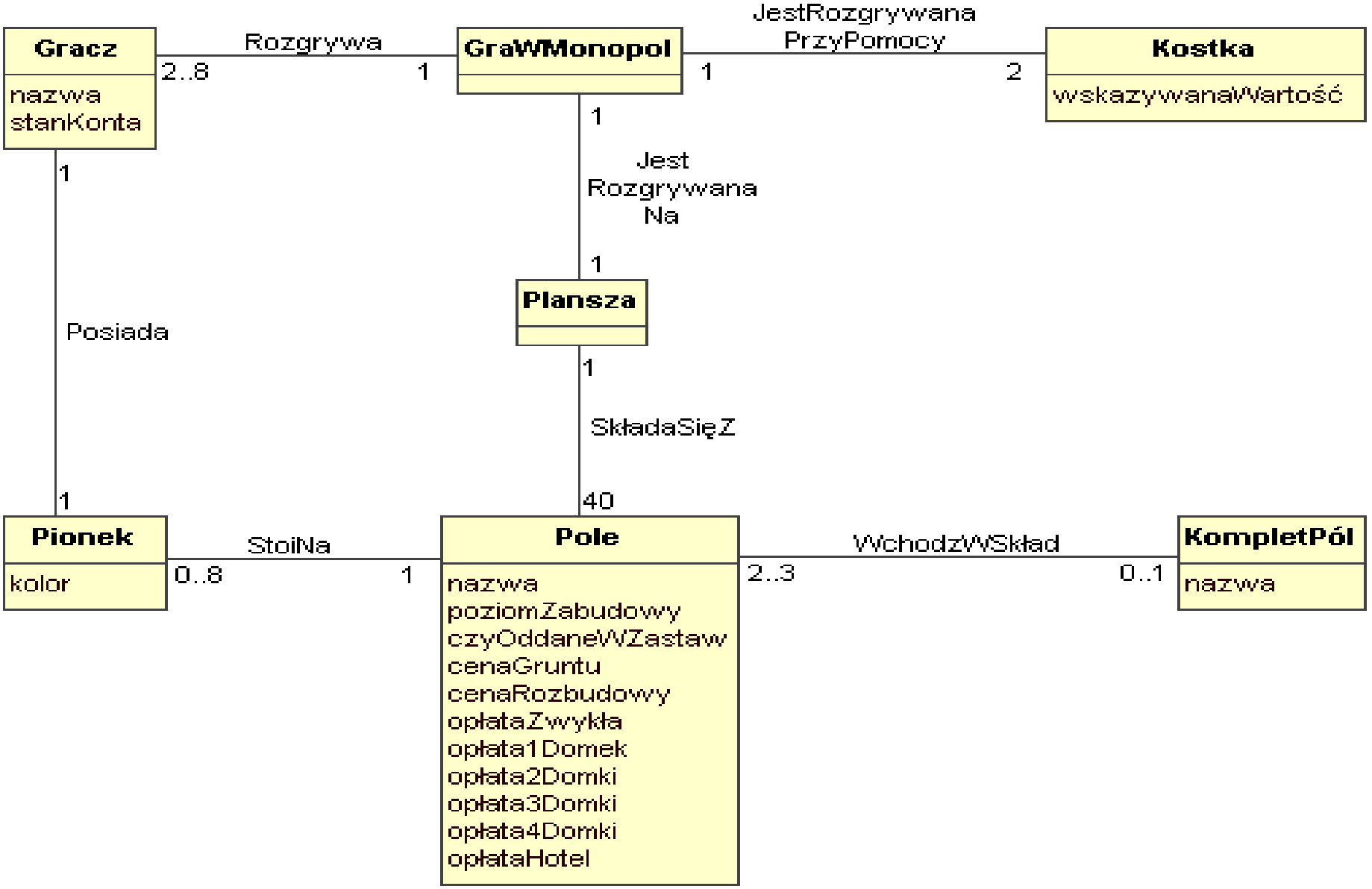
Klasy pojęciowe i powiązania dla gry w Monopol



Dodawanie atrybutów

- Wartości atrybutów (ang. attirbute) opisują egzemplarze klas pojęciowych. Nie wszystkie klasy pojęciowe muszą mieć atrybuty. Dodawanie atrybutów jest w zasadzie prostsze niż odnajdywanie klas pojęciowych czy powiązań i zazwyczaj zostawia się je na koniec. Są jednak dwie pułapki.
- Po pierwsze, trzeba pilnować, żeby **wartości atrybutów rzeczywiście opisywały egzemplarze klas pojęciowych**. Może się zdarzyć, że coś, co wydaje nam się atrybutem klasy A, powinno tak naprawdę być atrybutem klasy B, a między A i B powinno być powiązanie. Przykładowo numer jest atrybutem Miejsca, a nie Biletu.
- Po drugie niech **atrybuty będą jedynie wartościami typów podstawowych – "czystymi danymi" – jak napisy, liczby, wartości logiczne czy daty**. W przeciwnym przypadku prawdopodobnie znowu lepiej dodać klasę pojęciową i ustanowić z nią powiązanie. Przykładowo kierownik nie jest atrybutem Kina. To oddzielna klasa pojęciowa sama posiadająca wiele atrybutów jak imię, nazwisko czy dataUrodzenia.

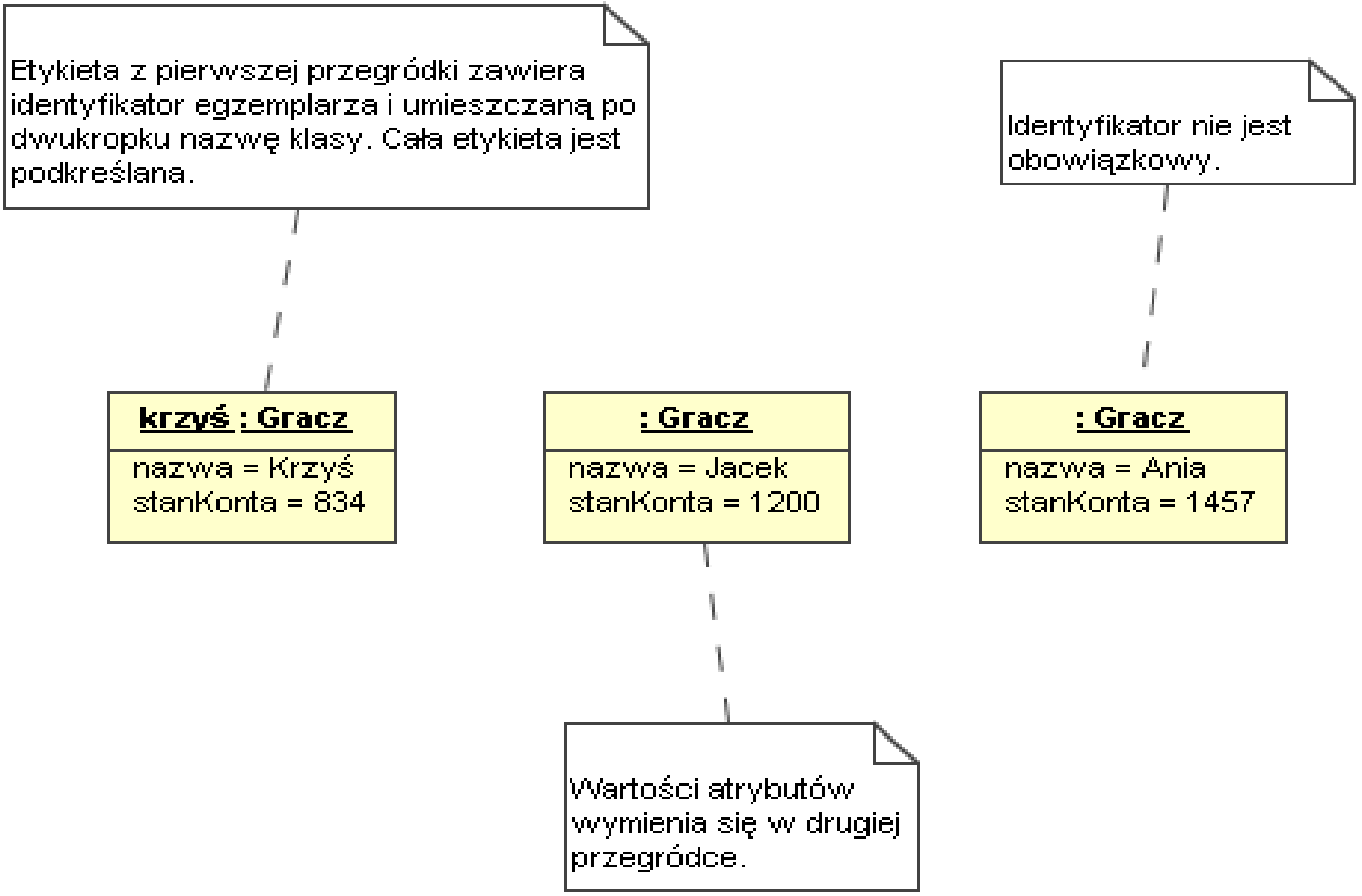
Częściowy model dziedziny dla gry w monopol



Obiekty i ich stan

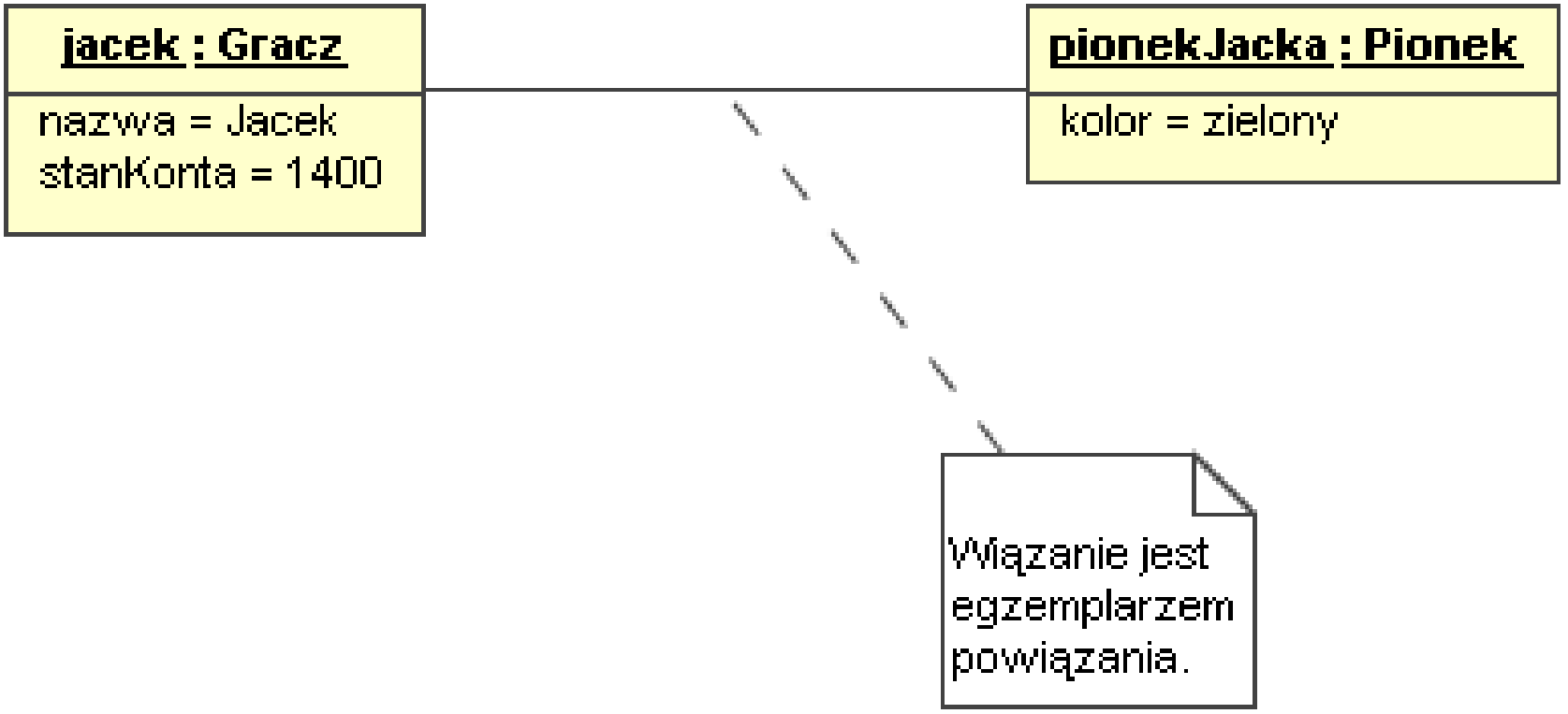
- Po rozpoczęciu tworzenia modelu dziedziny dla gry w Monopol można przyjrzeć się jakiejś rozgrywanej grze i wypisać wszystkie obiekty wraz z wartościami ich atrybutów.
- Wartości atrybutów pojedynczego obiektu określają jego stan. A stany wszystkich obiektów składają się w sumie na stan gry.
- Dotychczas na diagramach pokazywaliśmy jedynie klasy. Używając tej samej notacji można pokazywać poszczególne obiekty wraz z ich stanem.

Obiekty i ich stan



Wiązania

- Godne uwagi związki pomiędzy obiektami pokazujemy w ten sam sposób co powiązania między klasami. Przyjęło się je nazywać wiązaniami (ang. link). Tak samo jak obiekty są egzemplarzami klas, wiązania można uznać za egzemplarze powiązań. Poniższy diagram pokazuje wiązanie między obiektami klas Gracz i Pionek.



W jaki sposób konstruujemy program?

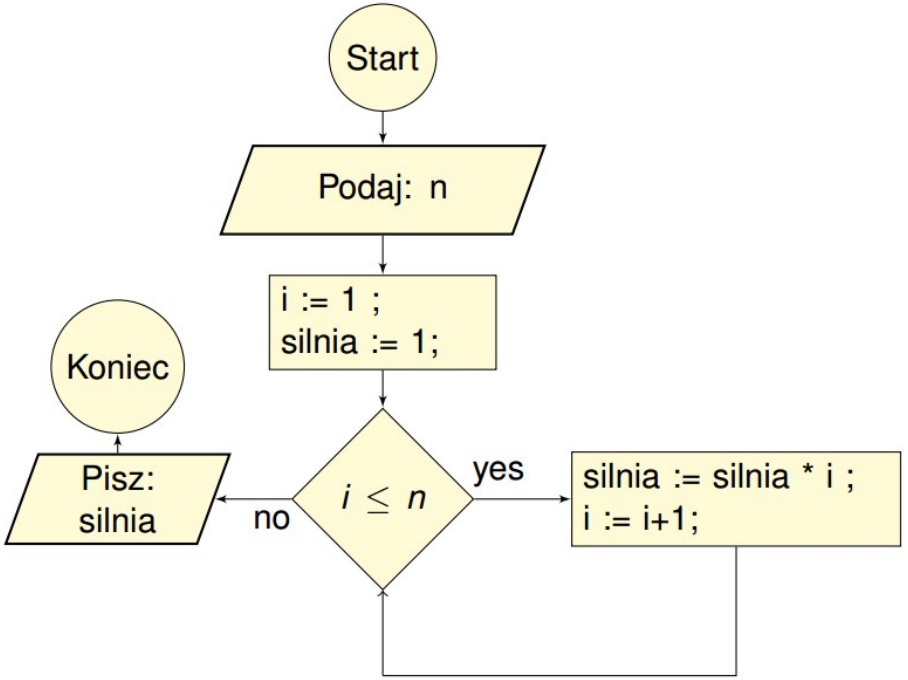
Algorytm

- Algorytm to (…) „przepis na rozwiązanie określonego zadania podany jako zbiór kolejnych poleceń; jego wykonawcą może być człowiek lub urządzenie automatyczne (np. komputer); (…) zapisany w jakimś języku programowania – program”

Encyklopedia PWN

Przykłady reprezentacji algorytmu

Schemat blokowy



Pseudokod

```
begin
  write("Podaj liczbe n = ");
  read (n);
  i:= 1;
  silnia := 1;
  while (i <= n) do
  begin
    silnia := silnia * i;
    i := i+1;
  end;
  write(silnia);
end.
```

Lista kroków

- **Dane:** dowolne nieujemne liczby rzeczywiste *a*, *b*, *c*
- **Wynik:** wartość średniej arytmetycznej liczb *a*, *b*, *c* równa *Srednia*
- **Krok 0:** Rozpocznij algorytm.
- **Krok 1:** Wprowadź wartości trzech liczb: *a*, *b*, *c*.
- **Krok 2:** Oblicz wartość wyrażenia: $Suma := a + b + c$.
- **Krok 3:** Oblicz wartość wyrażenia: $Srednia := Suma/3$.
- **Krok 4**: Wyprowadź wynik: *Srednia*.
- **Krok 5:** Zakończ algorytm.

Opis słowny

Dodanie lub mnożenie dwóch liczb
- sformułowanie zadania: oblicz sumę dwóch liczb naturalnych a i b, wynik oznacz przez S
- dane wejściowe: dwie liczby a i b
- cel obliczeń: oblliczenie sumy S=a+b
- ograniczenia: sprawdzenie warunku dla danych wejściowych, np. czy a i b są liczbami naturalnymi

Organizacja kodu źródłowego

Zapis instrukcji w postaci sekwencji

Organizacja kodu źródłowego

Programowanie strukturalne

```
Liczba1=12
Liczba2=100
Funkcja ObliczSume(a,b)

Funkcja ObliczSrednia(a[])

Procedura PokazWynik()

START
  Liczba1=100
  x=ObliczSume(Liczba1,Liczba2)
  y=ObliczSume(1,2)
  s=ObliczSrednia([x,y])
  PokazWynik()
STOP
```

Organizacja kodu źródłowego

Programowanie obiektowe

```
  Klasa: OBLICZENIA
    Liczba1=12
    Liczba2=100
    Wynik=0
    Metoda ObliczSume(a,b)
    Metoda ObliczSrednia(a[])
    Metoda PokazWynik()
START
  obliczenie1 : OBLICZENIA
  obliczenie2 : OBLICZENIA
  obliczenie1.Liczba1=100
  obliczenie1.ObliczSume(12,23)
  obliczenie2.ObliczSume(33,44)
  obliczenie2.ObliczSrednia([2,3,4])
  Obliczenie2.PokazWynik()
  Obliczenie1.PokazWynik()
STOP
```

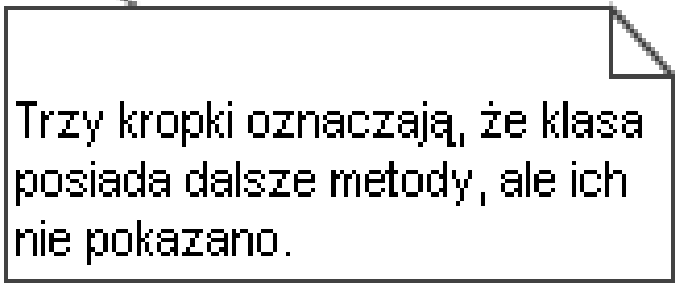
Projektowanie obiektowe

- Jak dotąd omówiliśmy, czym są klasy pojęciowe i jak je odnajdywać.
- Rozpoznanie klas pojęciowych pozwala lepiej zrozumieć dziedzinę problemu.
- Teraz pokażemy, że myślenie obiektowe ułatwia tworzenie programów i wyjaśnimy, co wyróżnia obiektowe języki programowania.
- Pokażemy również, że odnajdywanie klas pojęciowych nie było jedynie ćwiczeniem. Będzie się można na nich wzorować podczas projektowania i tworzenia programu.

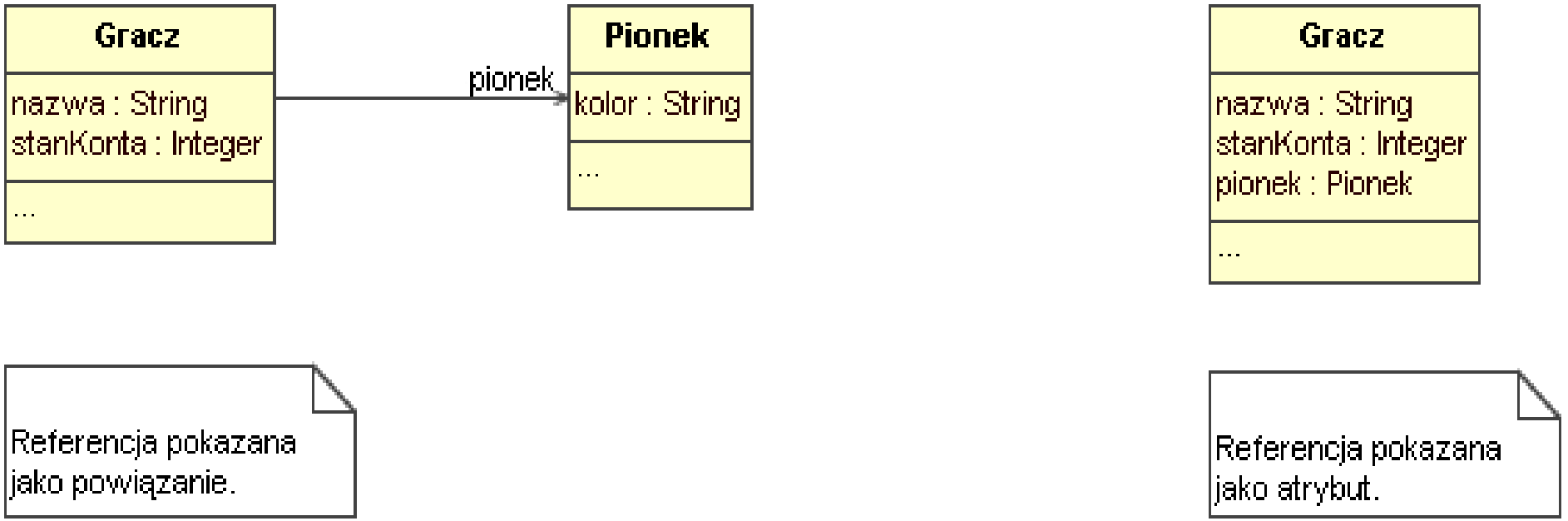
Obiekty a programowanie

Mimo, że na niniejszym diagramie użyliśmy UML, nie jest to już model dziedziny. Zaczęliśmy projektować oprogramowanie. Wiemy mniej więcej, jakie informacje będą przechowywane w poszczególnych klasach. Trzeba określić typy ich atrybutów, zdecydować jak zrealizować powiązania oraz rozdzielić metody.

Po podjęciu tych decyzji i umieszczeniu ich na diagramie nie mamy już do czynienia z klasami obiektowymi tylko projektowymi (design class)



Co z powiązaniami?



Ponieważ klasy projektowe mają być zaimplementowane, nie wystarczy wiedza, że występują między nimi powiązania. Trzeba zdecydować, jak te powiązania zrealizować. Zazwyczaj jedna z klas uczestniczących w powiązaniu będzie posiadała atrybut, na który zostanie przypisany obiekt drugiej klasy. Taki atrybut nazywa się referencją (ang. reference).

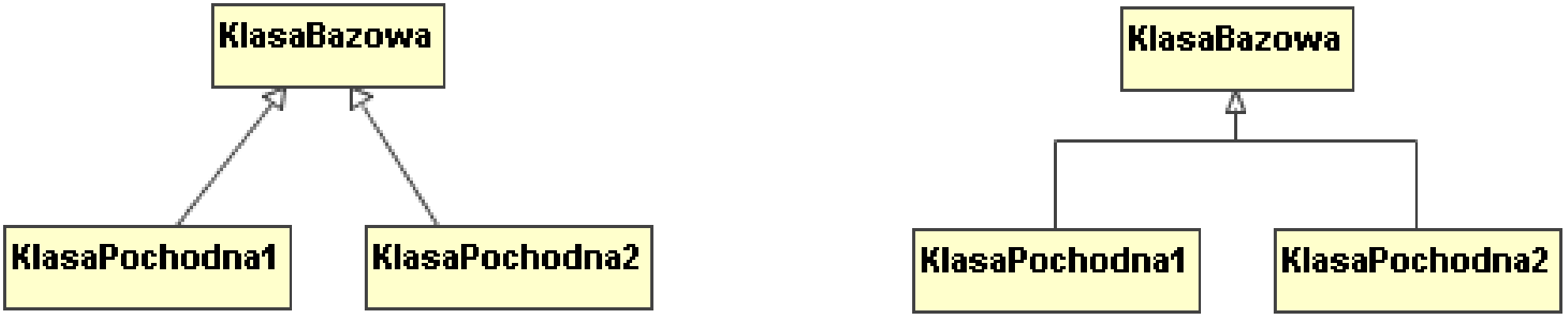
Zwróćmy uwagę, że również nazwa oraz stanKonta są referencjami. Klasy String i Integer są standardowo używane do przechowywania odpowiednio napisów oraz liczb całkowitych.

Pamiętaj, że sposób realizacji powiązania jest decyzją projektową. Nie należy podejmować decyzji projektowych już podczas tworzenia modelu dziedziny.

Typy podstawowe

- W większości obiektowych języków programowania są również nieobiektowe typy danych. Nazywa się je typami podstawowymi lub prostymi (ang. primitive type lub basic type).
- Typy podstawowe służą do przechowywania pojedynczych znaków, liczb całkowitych i rzeczywistych różnej precyzji oraz wartości logicznych.
- Są dodawane do języków obiektowych z dwóch powodów. Po pierwsze, ich wartości naturalnie nadają się do używania w wyrażeniach, a wypadku obiektów trzeba się posługiwać metodami.
- Po drugie, posługiwanie się nimi jest w wielu sytuacjach bardziej efektywne.

Hierarchie klas

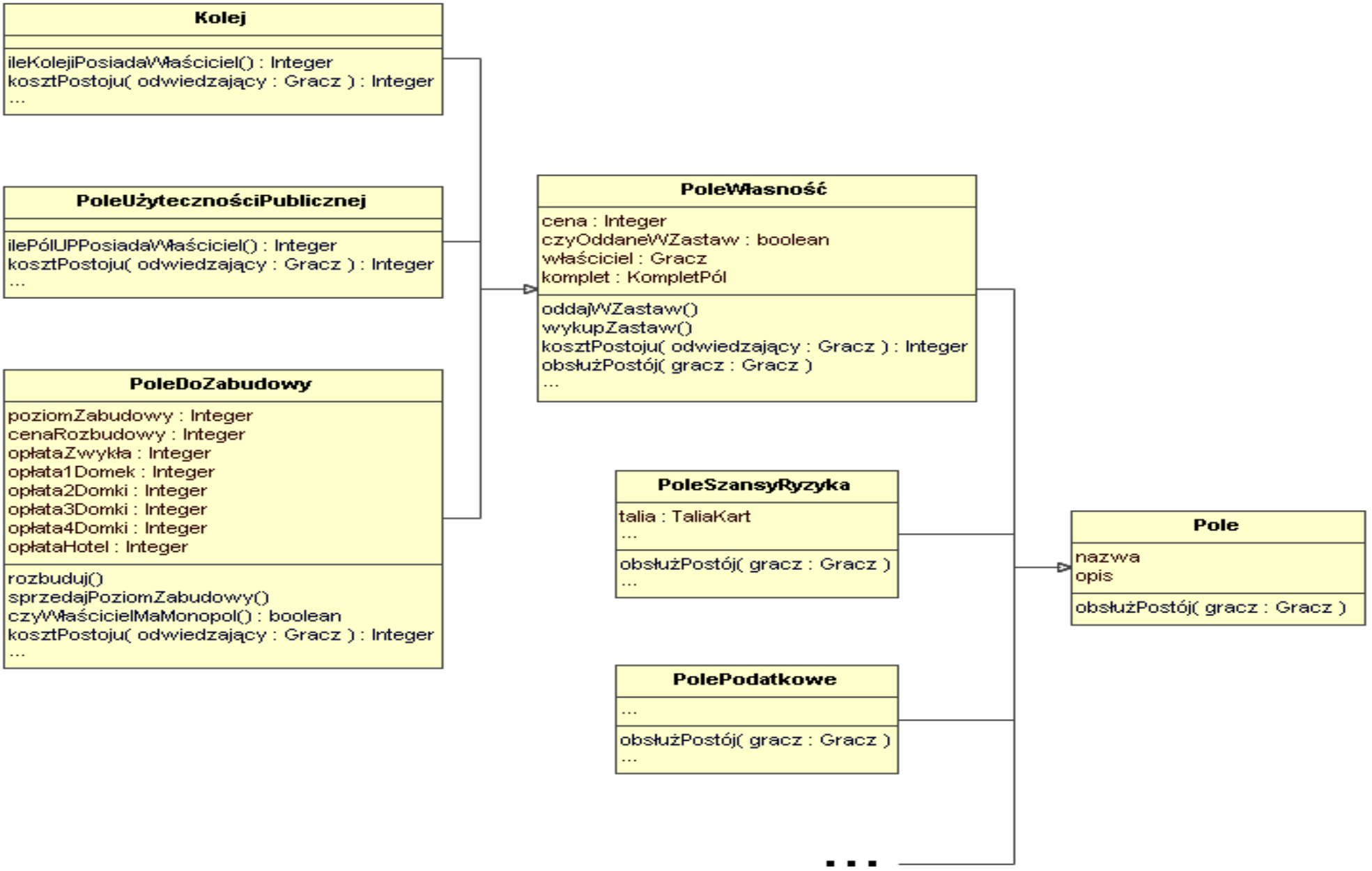


Dwa równoważne sposoby pokazywania hierarchii klas.

Dziedziczenie

- Aby nie powielać kodu, klasa pochodna dziedziczy (ang. inherits) wszystkie składowe klasy bazowej.
- Jeżeli zachowanie niektórych metod powinno być zmienione, klasa pochodna może je przedefiniować (ang. override) podając nowe definicje.
- Ponieważ klasa pochodna może również zawierać nowe składowe, często mówi się również, że podklasa rozszerza (ang. extend) nadklasę.

Hierarchia klas gry w monopol



Polimorfizm - wielopostaciowość

- Polimorfizm odnosi się do zadań, które można realizować na różne sposoby, np. licząc pole figury można posłużyć się jej wymiarami, albo współrzędnymi wierzchołków.
- W programowaniu polimorfizm można realizować na poziomie funkcji lub metod stosując techniki:
  - Przeciążanie (overload)
  - Przesłanianie (override)

Przeciążanie metod lub funkcji (overload)

```
Funkcja ObliczPole(a,b)

Funkcja ObliczPole(xy[])
```

Definiując w ten sposób funkcje lub metody o tej samej nazwie z różnymi wejściami realizujemy **polimorfizm statyczny** (compile-time polymorphism)

Przesłanianie metod (override)

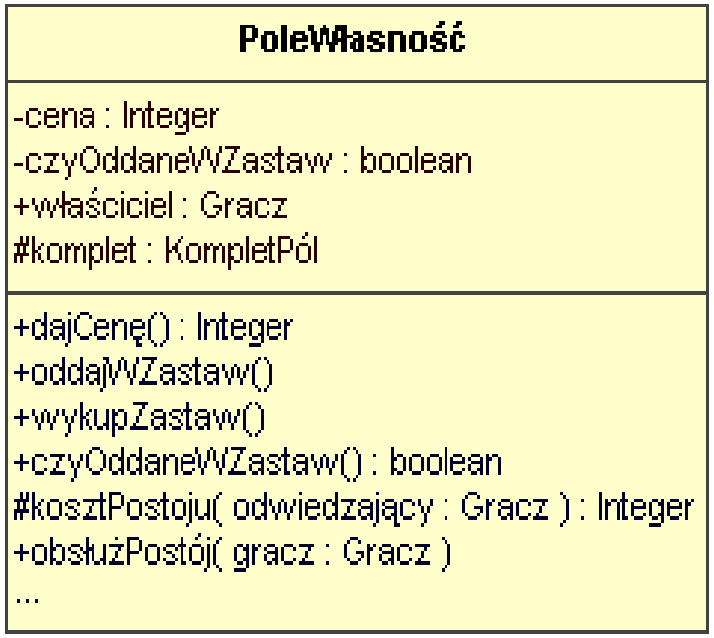
```
Klasa prostokat:
  metoda ObliczPole(a,b)
  metoda ObliczObwod

Klasa kwadrat(prostokat):
  metoda ObliczPole(a)
```

Definiując ponownie metodę ObliczPole() w klasie dziedziczącej sprawiamy, że wersja metody z nadklasy zostanie zastąpiona wersją w klasie dziedziczącej w ten sposób realizujemy **polimorfizm dynamiczny** (run-time polymorphism)

Kontrakty i widoczność

- Zlecenie odpowiedzialności obiektom za wykonanie konkretnych zadań można porównać do kontraktu z pracownikiem.
- Widoczność (visibility). Składowe klasy mogą być oznaczone jako:



(+) **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.

Hermetyzacja (enkapsulacja)

- Zwróćmy uwagę, że atrybuty cena i czyOddaneWZastaw w klasie PoleWłasność są prywatne.
- Do odczytywania i zmieniania ich wartości służą odpowiednio metody dajCenę() oraz oddajWZastaw(), wykupZastaw() i czyOddaneWZastaw().
- Dzięki takiemu rozwiązaniu obiekty innych klas nie mogą zmieniać ceny pola, a jedynie ją odczytywać (cena prawdopodobnie jest ustawiana w jakiejś niepokazanej tu metodzie inicjującej).
- Dla pola czyOddaneWZastaw istnieją metody zarówno przekazujące jego wartość, jak i ją zmieniające. Jednak w tym przypadku przy każdej zmianie wartości tego pola wykonywane są dodatkowe czynności, jak zmiana stanu konta jego właściciela.
- Uczynienie pola prywatnym wymusza używanie tych metod i gwarantuje, że czynności dodatkowe nie zostaną pominięte.

Konwencja get, set, is

- Z hermetyzacją wiąże się pewna konwencja. Jeżeli atrybut ma nazwę jakaśNazwa to metoda, która odczytuje jego wartość powinna się nazywać getJakaśNazwa() a metoda, która pozwala tą wartość zmienić setJakaśNazwa().
- W wypadku atrybutów przechowujących wartości logiczne metoda odczytująca wartość jest też czasami nazywana isJakaśNazwa().
- Nie istnieją ogólnie przyjęte polskie odpowiedniki tych przedrostków. Dobrym pomysłem wydaje się stosowanie oryginalnych przedrostków z atrybutami nazywanymi po polsku. Dodatkowym powodem, żeby tak robić jest fakt, że wiele bibliotek oraz narzędzi wspomagających tworzenie kodu zakłada stosowanie tej konwencji.

Przyczyny stosowania hermetyzacji

Można wyróżnić trzy główne powody wprowadzenia hermetyzacji do programowania obiektowego:

1)Wyodrębnia interfejs

2)Uodpornia tworzony model na błędy

3)Lepiej odzwierciedla rzeczywistość

Dziękuję za uwagę