Programowanie i projektowanie obiektowe

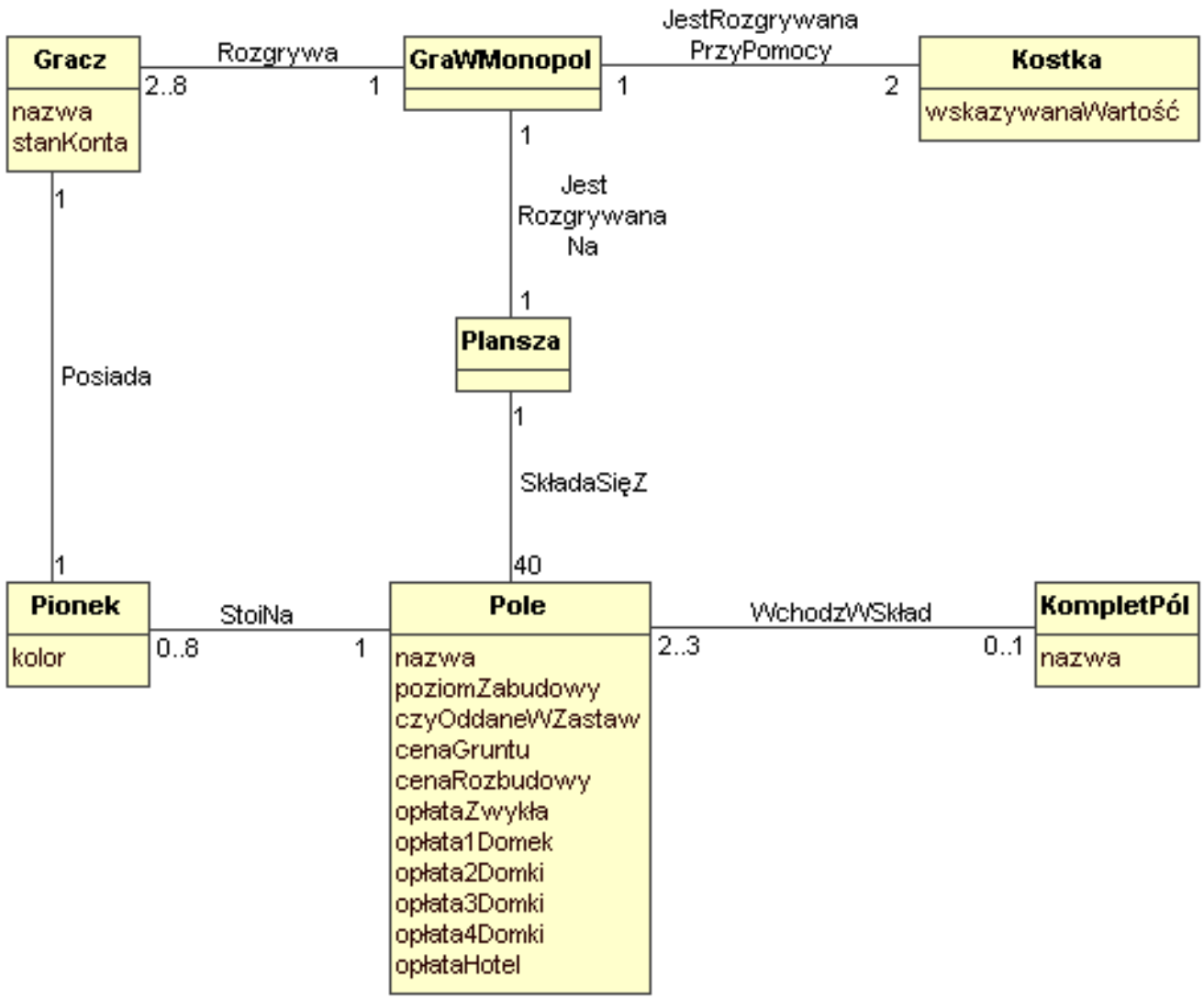
Projektowanie obiektowe

Paweł Daniluk

Wydział Fizyki

Jesień 2015

W poprzednim odcinku...



Obiektowy model dziedziny

Obiektowy model dziedziny identyfikuje klasy pojęciowe, ich atrybuty i powiązania pomiędzy nimi. Stanowi bazę do tworzenia projektu obiektowego.

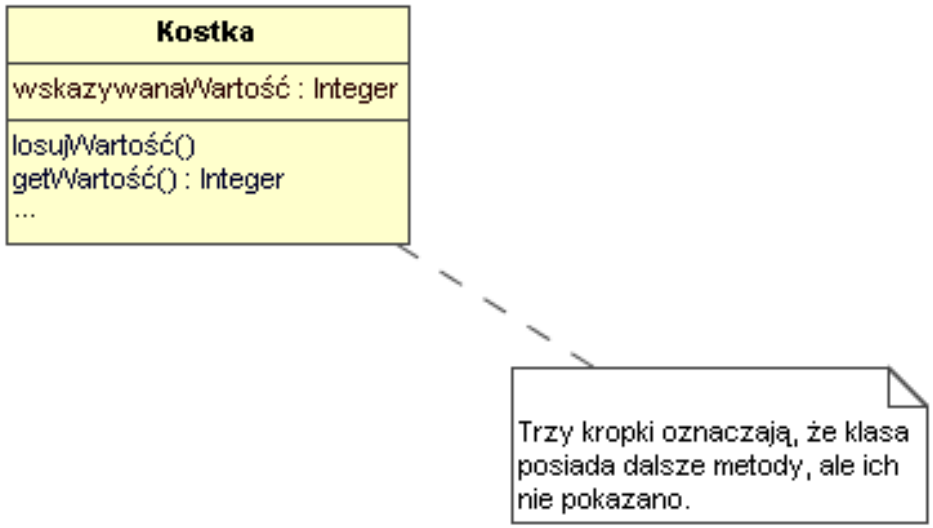
Projekt obiektowy

Zawiera

- klasy
- dziedziczenie
- atrybuty
- metody
- powiązania (wraz ze sposobem realizacji)

Metody

Programowanie obiektowe polega na wyznaczaniu obiektom odpowiedzialności. Te odpowiedzialności są dwóch typów: za pamiętanie pewnych danych i za wykonywanie operacji na tych danych. Przy czym zdecydowanie trudniejsze i ważniejsze jest rozdzielenie odpowiedzialności za wykonanie operacji.



Powiązania

Zazwyczaj powiązania realizuje się przy pomocy atrybutów.

Wybór klasy, do której należy atrybut określający powiązanie jest decyzją projektową.

Typy danych

Typy proste (podstawowe) (ang. *primitive types*, *basic types*)

Typy proste służą do przechowywania pojedynczych wartości – napisów, znakow, liczb, wartości logicznych. W naturalny sposób posługujemy się nimi w wyrażeniach. Są wydajniejsze.

Typy referencyjne (obiektowe)

Wartościami są referencje do obiektów (instancji klas).

Uwaga na przyszłość

Wartości typów prostych są niezmienialne – "A is A".
Wartości typów referencyjnych owszem – obiekty mają atrybuty.

Hierarchia klas

Klasa bazowa (nadklasa) (ang. *base class*, *superclass*)

Definuje składowe wspólne dla wszystkich wariantów.

Klasa pochodna (podklasa) (ang. *derived class*, *subclass*)

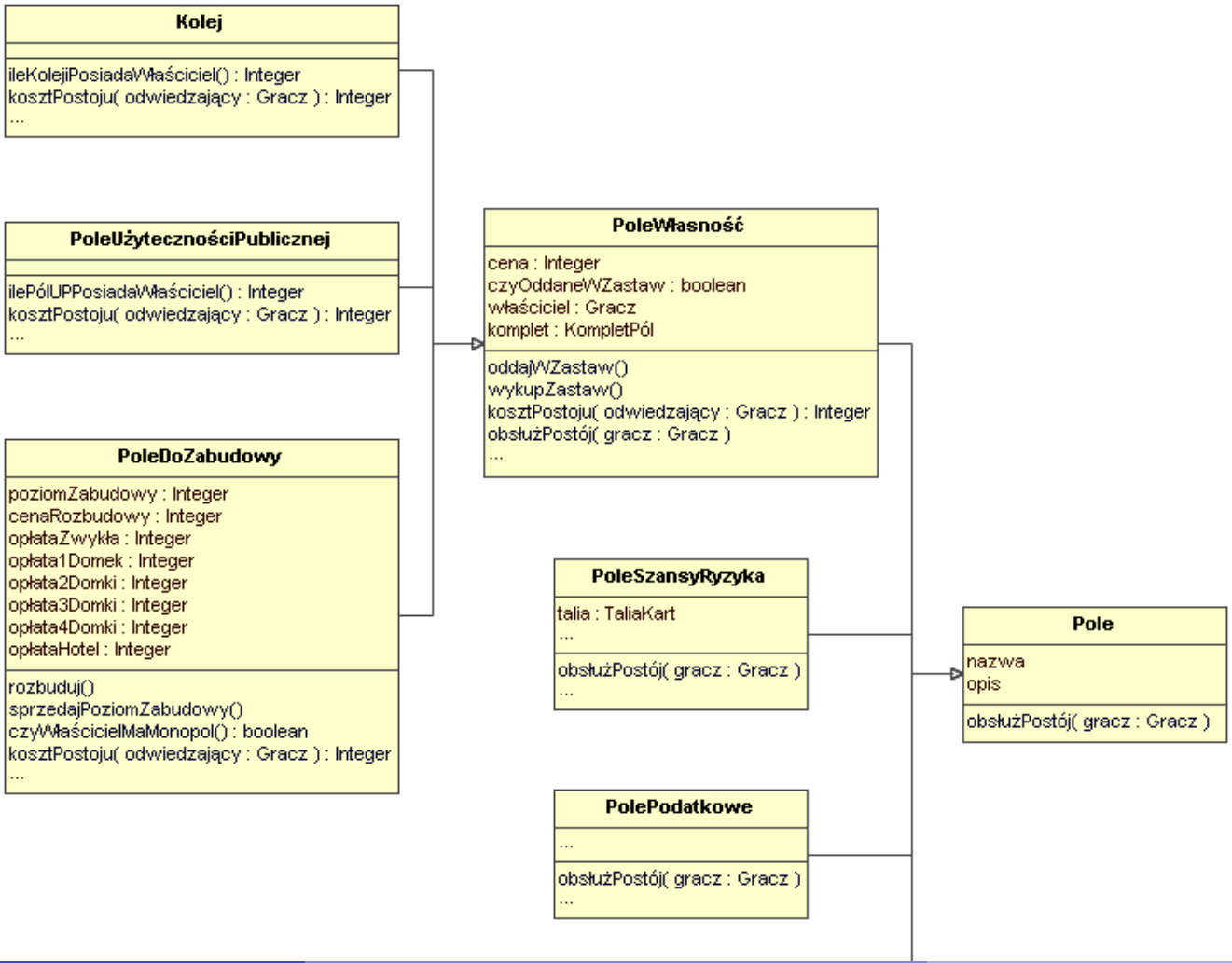
Definuje składowe charakterystyczne dla konkretnego wariantu.

Kryterium stosowalności

Podklasa jest szczególnym przypadkiem nadklasy.

Dziedziczenie

Podklasa dziedziczy (ang. *inherits*) składowe nadklasy.
Jeżeli jest to konieczne, może je przedefiniować (ang. *override*).



Odpowiedzialność

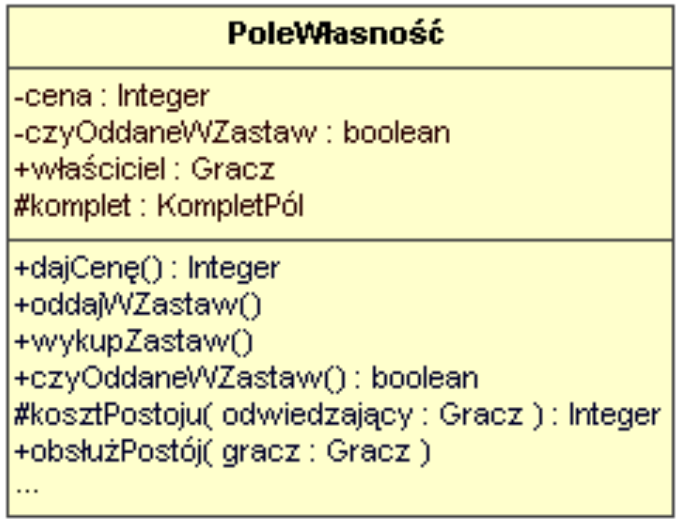
Projekt określa jakie składowe (funkcjonalności) klasa udostępnia. Do ich realizacji mogą być potrzebne składowe pomocnicze.

Składowe pomocnicze mogą zależeć od implementacji. Nie należy na nich polegać.

Widoczność

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



Kapsułkowanie

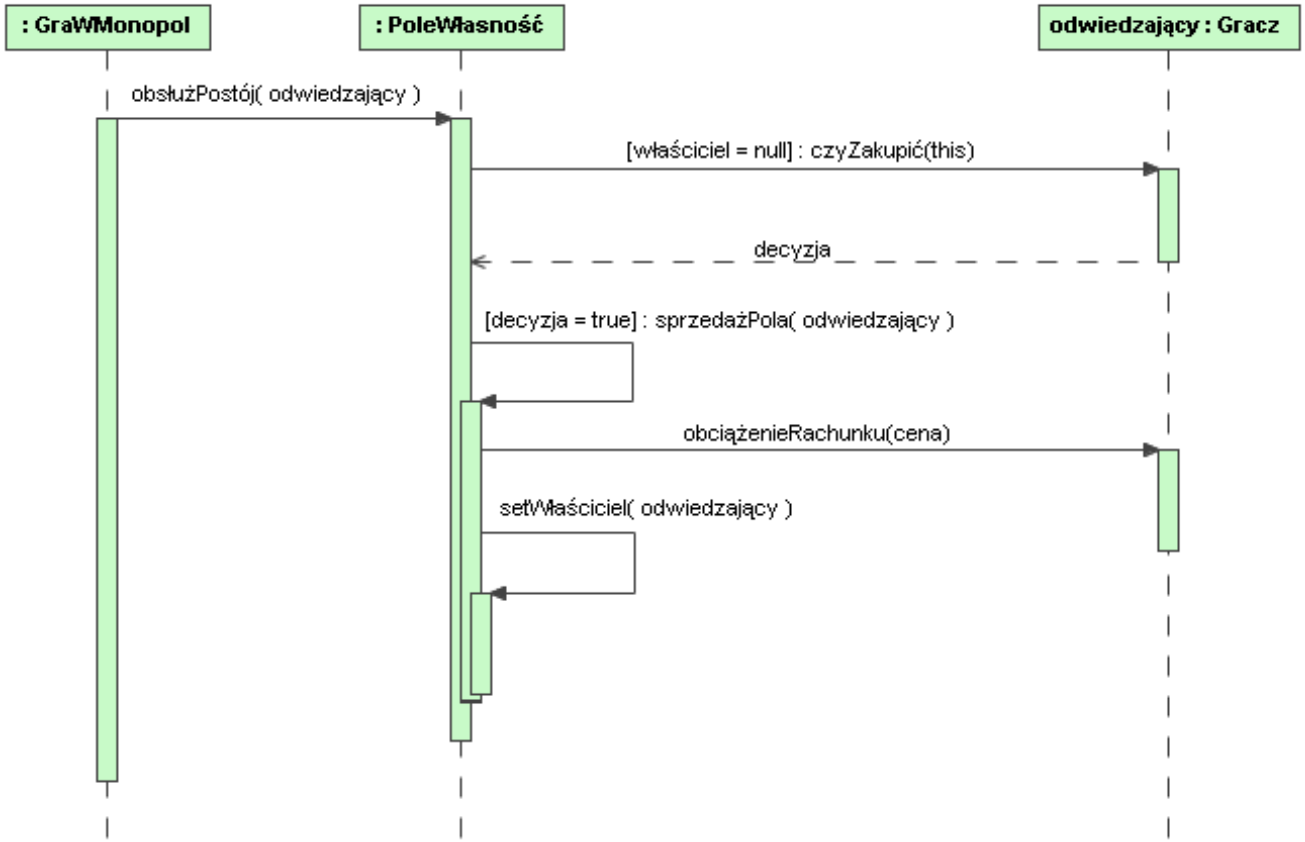
Oznaczać atrybutów jako prywatne chroni przed wieloma błędami, ale jak się wtedy do nich dostawać?

Konwencja *get*, *set* i *is*

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().

Diagramy przebiegu

Diagramy przebiegu służą do przedstawiania interakcji obiektów.



Sprzężenie, a spójność

Luźne sprzężenie

Sprzężenie (ang. coupling) jest miarą jak bardzo obiekty, podsystemy lub systemy zależą od siebie nawzajem. Przykładowo obiekt wykonujący metodę innego obiektu jest z nim sprzężony. Tak samo podklasa jest sprzężona z nadklasą.

Sprzężenie obiektów utrudnia wprowadzanie zmian w kodzie.

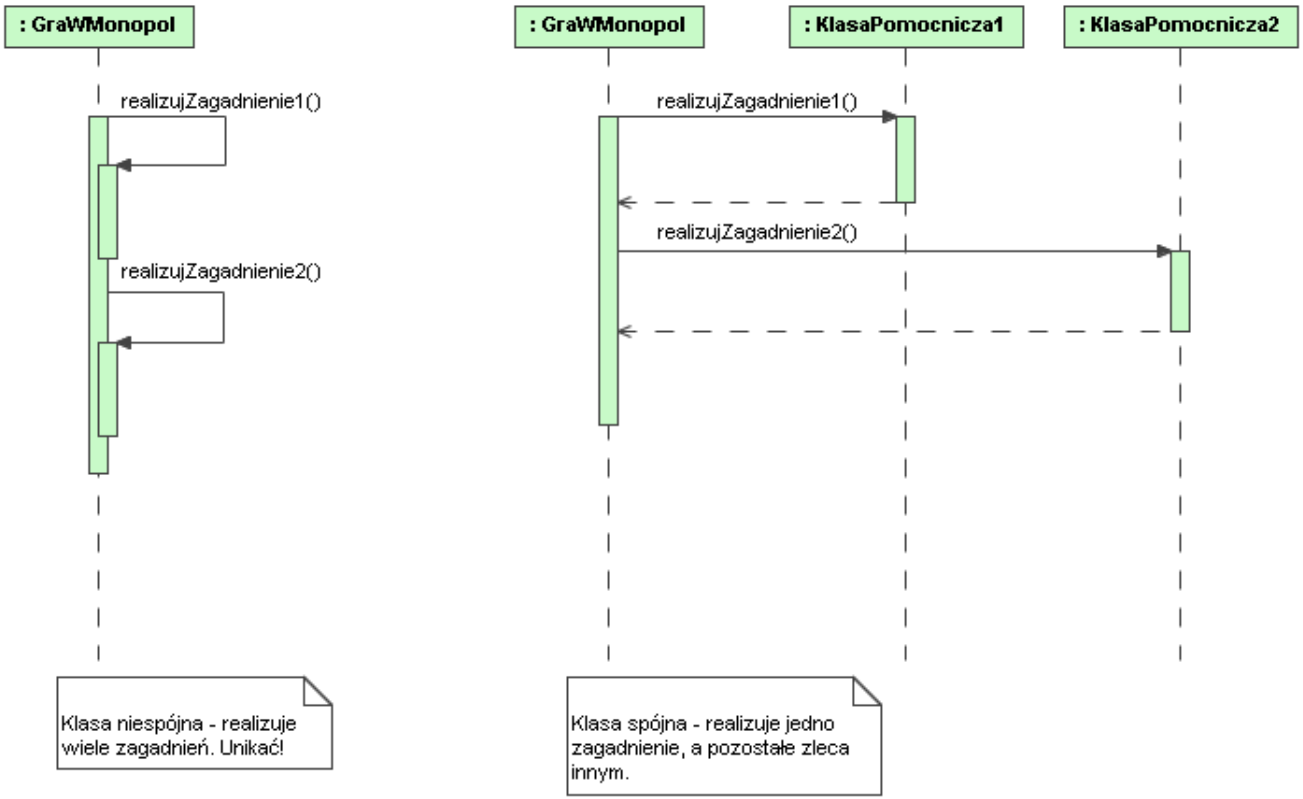
Wysoka spójność

Spójność to miara jak funkcjonalnie powiązane są metody danej klasy. Warto dbać o utrzymanie wysokiej spójności (ang. high cohesion).

Może się wydawać, że zachowanie wysokiej spójności przez zlecanie odpowiedzialności ma negatywny wpływ na sprzężenie. Jednak klasy mające wiele odpowiedzialności są zazwyczaj sprzężone z wieloma innymi klasami.

Klasy pomocnicze

Tworzenie klas pomocniczych pomaga zachować spójność.



Wzorce projektowe (ang. *design patterns*)

Zazwyczaj istnieje wiele rozwiązań projektowych danego problemu.

Kryteria jakości

- prostota
- pracochłonność
- elastyczność
- łatwość w utrzymaniu

Wzorce projektowe stanowią dobrze znane pary problem-rozwiązanie wraz z zaleceniami odnośnie stosowania, analizą zalet i wad, sposobami implementacji etc.

Wzorce projektowe (ang. *design patterns*) c.d.

Przykłady

Name	Description
Abstract factory	Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
Builder	Separate the construction of a complex object from its representation allowing the same construction process to create various representations.
Factory method	Define an interface for creating a single object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
Object pool	Avoid expensive acquisition and release of resources by recycling objects that are no longer in use. Can be considered a generalisation of connection pool and thread pool patterns.
Prototype	Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
Adapter or Wrapper	Convert the interface of a class into another interface clients expect. An adapter lets classes work together that could not otherwise because of incompatible interfaces.
Composite	Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.
Facade	Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
Module	Group several related elements, such as classes, singletons, methods, globally used, into a single conceptual entity.
Iterator	Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Null object	Avoid null references by providing a default object.
Observer or Publish/subscribe	Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
Servant	Define common functionality for a group of classes
Strategy	Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.