

# Technologie obiektowe

materiały wykładowe

*(c) 1997–2001 Marek Kisiel-Dorohinicki*

## Literatura

- Peter Coad, Edward Yourdon  
„Analiza obiektowa”
- Kazimierz Subieta  
„Obiektość w projektowaniu i bazach danych”
- Andrzej Jaszkievicz  
„Inżynieria oprogramowania”
- Mariusz Kliszewski  
„Inżynieria oprogramowania obiektowego – Analiza obiektowa”
- James Martin, James J. Odell  
„Podstawy metod obiektowych”

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Część I

## Wprowadzenie

## Inżynieria oprogramowania – rys historyczny

- bardzo gwałtowny rozwój technik komputerowych
- kryzys oprogramowania
  - a) duża złożoność systemów
  - b) niepowtarzalność i specjalizacja przedsięwzięć
  - c) pozorna łatwość tworzenia kodu
  - d) nieprzejrzystość procesu tworzenia
- ewolucja technologii tworzenia systemów komputerowych
  - potrzeba projektowania, modelowania i planowania
  - podejście strukturalne w programowaniu, projektowaniu i analizie

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Zakres inżynierii oprogramowania

prorowadzenie przedsięwzięć informatycznych:

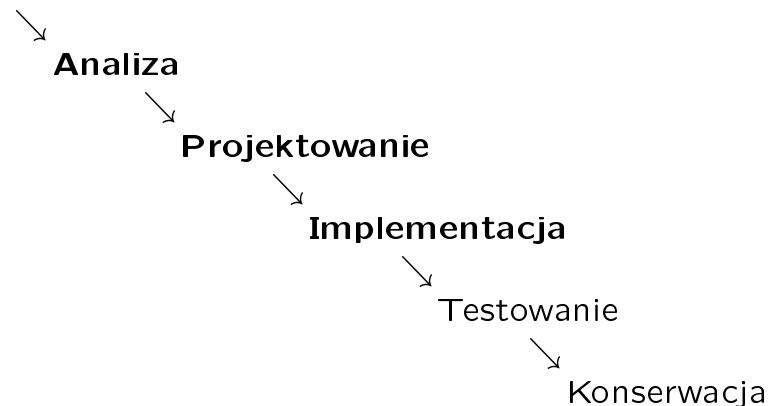
- techniki planowania, harmonogramowania i monitorowania
- metody analizy i projektowania
- sposoby zwiększania niezawodności, metody testowania, procedury kontroli jakości
- przygotowywanie dokumentacji technicznej i użytkowej
- konserwacja oprogramowania: usuwanie błędów, wprowadzanie modyfikacji i rozszerzeń
- techniki pracy zespołowej
- narzędzia CASE

etc.

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Etapy budowy systemu komputerowego

Faza strategiczna



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Podstawowe pojęcia związane z analizą

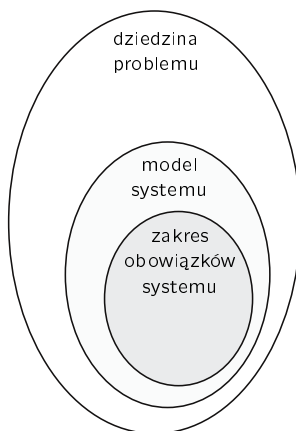
**problem** – kwestia do rozwiązania lub rozważenia

**dziedzina problemu** – rozważane pole działania

**system** – powiązany w jedną całość zbiór elementów

**zakres obowiązków systemu** – fragment dziedziny objęty wymaganiami wobec systemu

**analiza** – studium dziedziny problemu prowadzące do specyfikacji modelu systemu w kontekście jego obowiązków



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Rola modelowania w analizie

**model** = uproszczona reprezentacja czegoś

- odzwierciedla zdolność do abstrakcji
- pomaga lepiej zrozumieć złożoność dziedziny lub systemu
- pozwala na wizualizację, specyfikację i dokumentację systemu

zasady modelowania:

1. wybór modeli ma decydujące znaczenie na sposób podejścia do problemu i jego rozwiązania
2. najlepsze modele dobrze odzwierciedlają rzeczywistość
3. każdy model może być wyrażony na różnych poziomach szczegółowości
4. pojedynczy model rzadko jest wystarczający – najlepsze zrozumienie daje możliwie mała liczba możliwie niezależnych modeli

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Projektowanie a analiza

- analiza koncentruje się na *problemie*, a projektowanie na *rozwiązaniu*
- celem analizy jest budowa modelu *dziedziny*, zaś celem projektowania – modelu *systemu* dla konkretnej platformy, narzędzi i sprzętu
- w pewnym sensie analiza specyfikuje model systemu niejako „z zewnątrz”, a projektowanie zajmuje się szczegółami jego architektury „wewnątrznej”
- silny rozdział między etapami analizy i projektowania spowodowany jest różnicami w tradycyjnie stosowanych metodologiach

## Proces tworzenia oprogramowania

- model kaskadowy (ang. *waterfall model*)  
analiza → projektowanie → programowanie → testowanie
- realizacja sterowana dokumentami (ang. *document-driven realisation*)
- prototypowanie (ang. *prototyping*)  
[budowa prototypu → testowanie] +
- realizacja przyrostowa (ang. *incremental development*)  
[budowa części systemu → testowanie] +
- programowanie ekstremalne (ang. *extreme programming*)

## Potencjalne problemy

### komunikacja

- wewnętrzna: różne grupy ludzi na różnych etapach
- zewnętrzna: zleceniodawca, eksperci, użytkownicy

### metodologie, standardy, narzędzia

różne na kolejnych etapach, zmienne w czasie, inaczej wyszkoleni ludzie

### zmienne wymagania

niezdecydowany klient, zmieniająca się sytuacja ekonomiczna i prawna, konkurencja, nowe technologie

### istniejące ograniczenia

technologia, harmonogram, budżet, ludzie

## Modelowanie obiektowe w inżynierii oprogramowania

- nie tylko programowanie, choć historycznie pierwsze
- różne metody w oparciu o podobne przesłanki
- stale rosnąca popularność
- nie jest środkiem uniwersalnym

## Paradygmat obiektowy

— opanowywanie złożoności (dziedziny) problemu opiera się na powszechnie (choć nie zawsze świadomie) stosowanych zasadach:

- tworzenie i nazywanie pojęć
- rozróżnianie poszczególnych obiektów
- znajdowanie zależności między pojęciami i wyprowadzanie nowych
- określanie relacji między obiektami

## Zalety podejścia obiektowego

- bezpośrednie odwzorowanie dziedziny (problemu) na system (rozwiązanie)
- spójność sposobu myślenia i reprezentacji między kolejnymi etapami, szczególnie między analizą a projektowaniem
- (większa) łatwość porozumienia dla fachowca i laika
- (większa) spójność i stabilność modelu
- możliwość wielokrotnego wykorzystania opracowanych rozwiązań
- (większa) łatwość modyfikacji i rozszerzania systemu

## Zasady modelowania

**abstrakcja** – ignorowanie tych aspektów przedmiotu, które nie są istotne z punktu widzenia bieżącego celu; pomijanie szczegółów, które nie są istotne w bieżącym kontekście

**hermetyzacja** – ukrywanie informacji o składowych systemu tak, aby zamykały one w sobie możliwie jedną decyzję projektową; zamykanie szczegółów implementacji za interfejsem zdefiniowanym w kategoriach dozwolonych usług

**skala** – taka organizacja systemu aby dowolnie duża składowa była przejrzysta dla obserwatora; rezygnacja ze szczegółów przy prezentowaniu całości złożonego problemu

## Podstawowe jednostki abstrakcji

**pojęcie** – idea lub wyobrażenie z dziedziny problemu posiadająca swoją nazwę i znaczenie

**obiekt** – element dziedziny problemu; jednostka z jednoznaczoną identyfikacją, posiadająca pewien stan i przejawiająca określone zachowanie; egzemplarz pojęcia

**klasa** – opis grupy obiektów o podobnych własnościach, relacjach i znaczeniu; implementacja pojęcia

**typ obiektowy** = pojęcie lub klasa

**abstrakcyjny typ danych** = klasa

## Własności obiektów

**stan** – abstrakcja dotychczasowej historii obiektu w aktualnym sposobie zachowania; sytuacja w trakcie życia obiektu, kiedy spełniony jest określony warunek

**atrybut** – nazwana cecha obiektu opisująca możliwe wartości jakie może przyjąć; porcja danych opisująca określoną cechę obiektów w danej klasie

**usługa** – nazwane zachowanie obiektu jakie jest on zobowiązany przejawiać; określona funkcjonalność jaką przejawiają obiekty danej klasy

**komunikat** – zlecenie wykonania usługi zawierające potrzebne ku temu informacje (parametry)

**własność** = atrybut lub usługa

## Sposoby organizacji

**dziedziczenie** – wyrażanie podobieństwa między elementami przy pomocy hierarchii uszczegółowiania i uogólniania; mechanizm pozwalający elementom bardziej specyficznym na wykorzystanie struktury i zachowania elementów bardziej ogólnych

**powiązanie** – relacja między obiektami dwóch klas posiadająca określone znaczenie

**agregacja** – specyficzna forma powiązania z wyróżnieniem obiektów całości i ich składowych (części)

## Część II

# Modelowanie obiektowe w analizie

## Zalety analizy obiektowej

1. Nacisk na zrozumienie dziedziny problemu → bardziej ambitne dziedziny zastosowań.
2. Metody organizacji wzorowane na myśleniu człowieka → lepsze zrozumienie użytkownika i eksperta.
3. Traktowanie danych (*atrybutów*) i procesów (*usług*) jako naturalnej całości → zwiększenie spójności analizy.
4. Jawna reprezentacja wspólnych cech (*dziedziczenie*) → uproszczenie modelu.
5. Ukrywanie cech nietrwałych (*hermetyzacja*) → specyfikacja poddająca się zmianom.
6. Modele odpowiadające rzeczywistości, takie same metody organizacji → wielokrotne wykorzystanie wyników.
7. Ciągłość reprezentacji w kolejnych etapach budowy systemu → łatwy powrót do etapu analizy w celu modyfikacji modelu.

## Źródła informacji

- własna obserwacja,
  - zdanie ekspertów w dziedzinie,
  - wyniki analizy i zrealizowane systemy w podobnych dziedzinach,
  - wszelkie dokumenty,
  - prototypy protokołów,
- etc.

## Modelowanie pojęć dziedziny

- pamiętane rzeczy i zdarzenia,
  - występujące miejsca,
  - odgrywane role,
  - procedury operacyjne,
  - jednostki organizacyjne,
  - inne systemy,
  - urządzenia,
- etc.

## Kryteria poprawności klasy

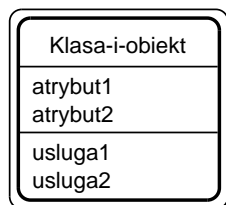
1. Pamiętane informacje – obecność atrybutów.
2. Wymagane zachowania – obecność usług.
3. Złożony stan – wielość atrybutów.
4. Więcej niż jeden obiekt w klasie lub jej specjalizacjach.
5. Zastosowanie wszystkich atrybutów i usług w obiektach klasy.
6. Spełnione wymagania dziedziny problemu.
7. Nie specyfikujemy rezultatów pochodnych.

## Część III

## Notacja i metodologia Coad/Yourdon

## klasa-i-obiekt

klasa i jej obiekty:

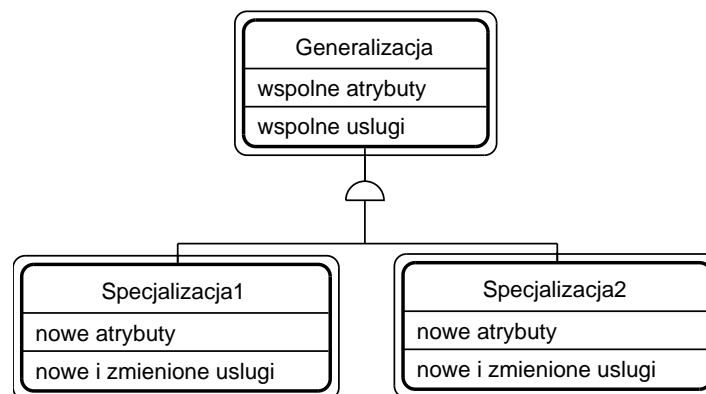


klasa bez obiektów:



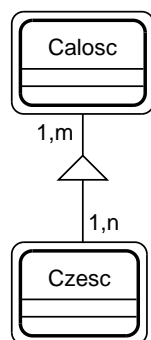
Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## struktura gen-spec



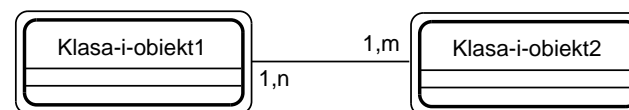
Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## struktura całość-część



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## powiązanie



## komunikat



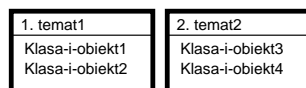
Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## tematy

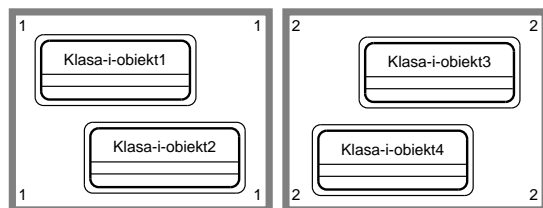
zwinęte:



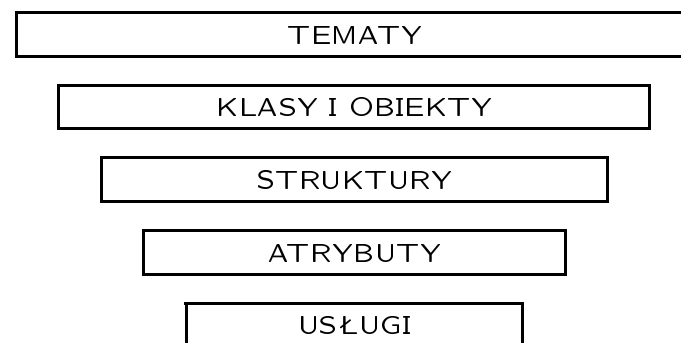
częściowo rozwinięte:



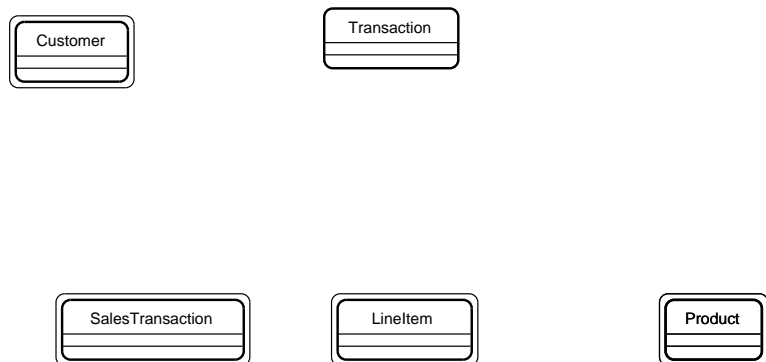
rozwinięte:



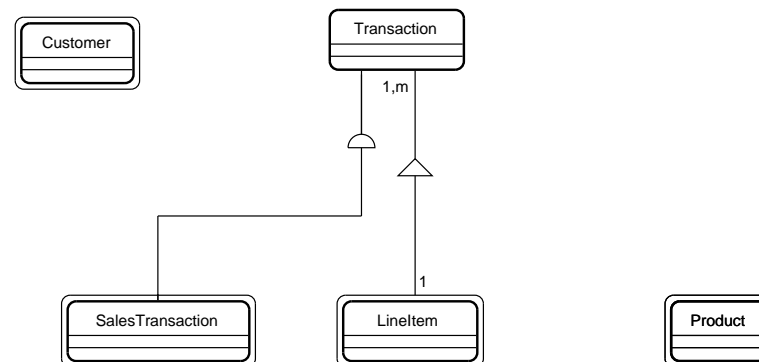
## Warstwy i czynności w analizie obiektowej



## klasy-i-objekty

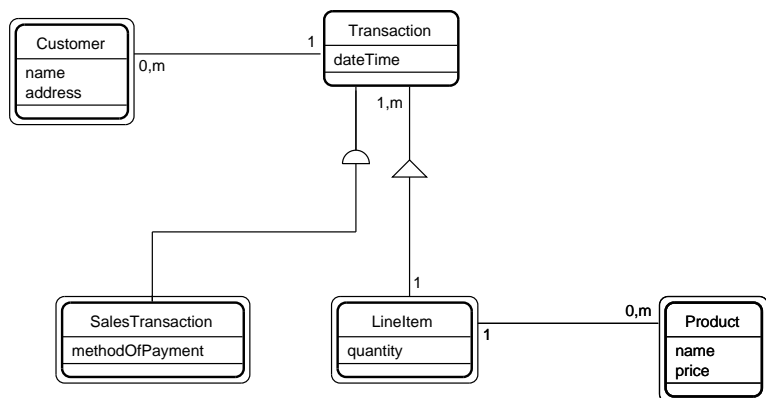


## klasy-i-objekty + struktury



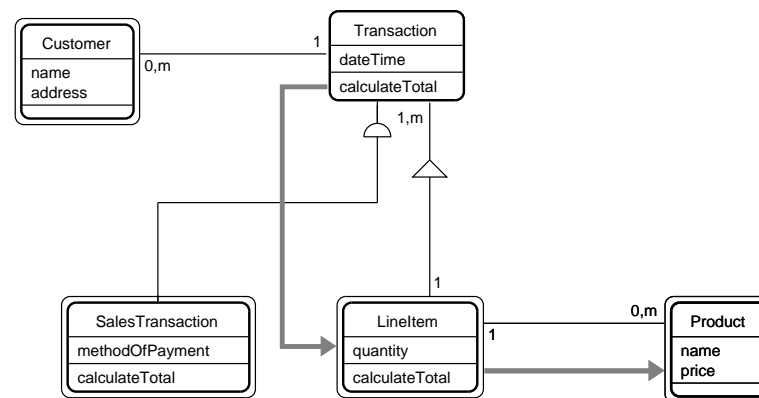


## klasy-i-objekty struktury atrybuty



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## klasy-i-objekty struktury atrybuty usługi



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## wyszukiwanie klas-i-objektów

**obiekt** – abstrakcja czegoś w dziedzinie problemu, odzwierciedla zdolności systemu do przechowywania o tym informacji (atrybuty) oraz wykonywania na tym operacji (usługi)

**klasa** – opis obiektu (obiektów) z jednolitym zbiorem atrybutów i usług oraz sposobu tworzenia nowego obiektu (obiektów)

**klasa-i-obiekt** – klasa i obiekty w tej klasie

### Metoda długiej i krótkiej listy

1. znajdujemy możliwie jak najwięcej elementów dziedziny – potencjalnych kandydatów na klasy-i-objekty → **długa lista**
2. długą listę weryfikujemy, usuwając zbędne (i ewentualnie uzupełniając brakujące) klasy-i-objekty → **krótka lista**

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## wyszukiwanie struktur

**struktura** – sposób organizacji

**gen-spec** – organizacja klas, oparta na dziedziczeniu

**całość-część** – organizacja obiektów, oparta na agregacji

### Strategie wyszukiwania

**struktura gen-spec**

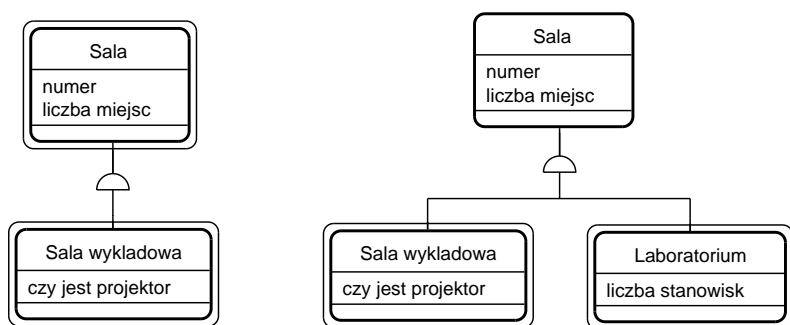
traktować każdą klasę jako generalizację i szukać specjalizacji (i odwrotnie) sprawdzając czy relacja ma sens w dziedzinie (zakresie obowiązków systemu) i czy występuje dziedziczenie

**struktura całość-część**

traktować obiekty każdej klasy jako całości i szukać części (i odwrotnie) sprawdzając czy relacja ma sens w dziedzinie (zakresie obowiązków systemu)

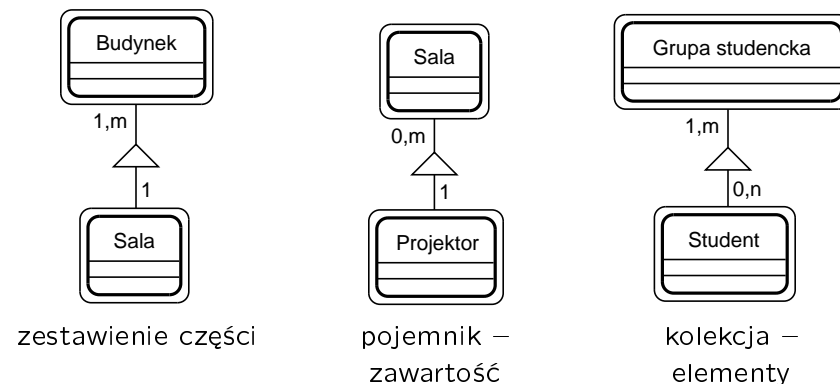
Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Przykłady struktur gen-spec



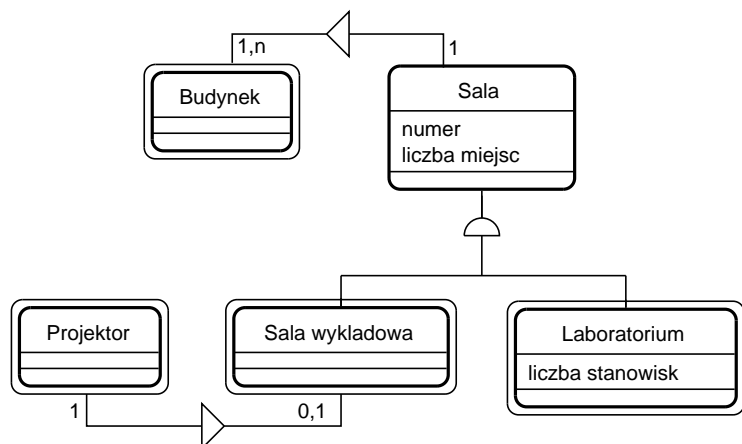
Copyright (c) 1997–2001 Marek Kisiel-Dorohiniński

## Przykłady struktur całość-część



Copyright (c) 1997–2001 Marek Kisiel-Dorohiniński

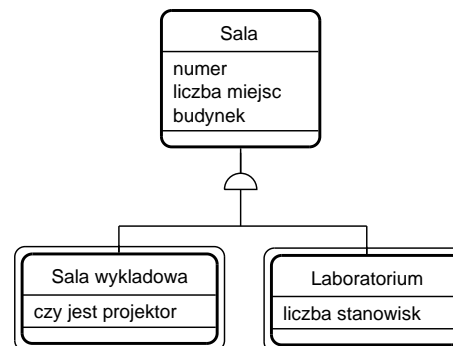
## Struktury wielokrotne



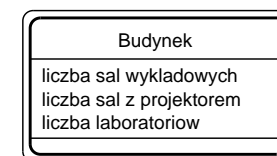
Copyright (c) 1997–2001 Marek Kisiel-Dorohiniński

## Struktury wielokrotne – cd.

może można prościej:



albo jeszcze prościej:



Copyright (c) 1997–2001 Marek Kisiel-Dorohiniński

## Wyszukiwanie tematów

**tematy** – części modelu odzwierciedlające poddziedziny problemu

### Strategia wyszukiwania

1. najwyższą klasę każdej struktury oraz klasy niezależne awansować na temat
2. łączyć tematy ze sobą według zagadnień i tak, aby minimalizować przecięcia struktur

uwaga: jedna klasa-i-obiekt może należeć do więcej niż jednego tematu!

## Identyfikacja atrybutów i powiązań

- model pełnej hermetyzacji atrybutów
- identyfikacja obiektu – ukryte ID
- identyfikacja powiązania – ukryte IDD

### Strategia wyszukiwania atrybutów

pytamy z punktu widzenia obiektu danej klasy:

- jak jest opisany?
- co musi wiedzieć?
- jakie informacje musi pamiętać?
- w jakim może być stanie?

## Strategia warstwy usług

### identyfikacja stanów

badamy potencjalne wartości atrybutów i sprawdzamy, czy zachowanie obiektu jest różne dla tych wartości

### identyfikacja usług

szukamy usług związanych z każdą zmianą stanu

### identyfikacja powiązań komunikatów

rozważamy zachowania obiektów:

- usługi jakich obiektów są mu potrzebne?
- jakie obiekty potrzebują jego usług?

### specyfikacja usług

- *usługi algorytmicznie proste* (np. pobierz, ustaw, powiąż, utwórz) – zbędna szczegółowa specyfikacja
- *usługi algorytmicznie złożone*, (np. oblicz, monitoruj) – schemat blokowy

IV

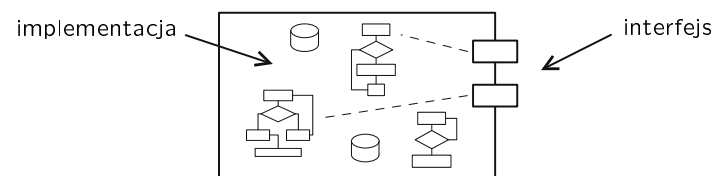
i i | i  
i

## Abstrakcja: upraszczanie złożoności

- podstawowe narzędzie modelowania, także obiektowego
- ma na celu redukcję (**pomijanie**) szczegółów, aby ułatwić zrozumienie złożoności modelu/systemu
- może dotyczyć różnych *aspektów* modelu/systemu: danych, funkcji, procesów, stanów, czy oczywiście obiektów
- może być stosowana na różnych *poziomach*: zgodnie z zasadą skali kolejne poziomy abstrakcji powinny prowadzić od dużych składowych systemu do coraz mniejszych, zwiększając ilość prezentowanych informacji

## Hermetyzacja

- podział modelu/systemu na składowe (**modularyzacja**) ułatwia modelowanie jego architektury: moduły stają się jednostkami abstrakcji
- hermetyzacja oznacza zamykanie struktury wewnętrznej składowej (**implementacji**) za dobrze zdefiniowanymi granicami (**interfejsem**)



## Obiekty

obiekt = identyfikacja + stan + zachowanie

- podstawowe jednostki abstrakcji modelu obiektowego
- mogą być **wskazywane**, a nie nazywane
- posiadają stan zapisany w wartościach atrybutów (inaczej niż **procedury** – usługi)
- są odróżnialne od innych obiektów posiadających taki sam stan oraz przejawiają złożone zachowania (inaczej niż **struktury danych** – atrybuty)

## Stany obiektu

- stan to określone **wewnętrzne** uwarunkowanie obiektu mające wpływ na jego **zachowanie**
- informacja o stanie obiektu zapisana jest w jego **atrybutach** i jest dostępna jedynie poprzez jego **usługi** (hermetyzacja)
- nowo utworzony obiekt musi się znaleźć w określonym poprawnym stanie, który jednakże może zależeć od sposobu wykonania odpowiedzialnej za to usługi
- stan obiektu zmienia się jedynie na skutek wykonania **usługi**, choć niektóre obiekty mogą spontanicznie (same z siebie) zmieniać swój stan – np. **aktor**, **aktywny obiekt**

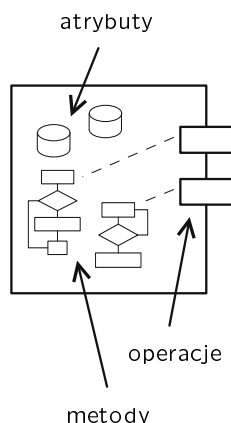
## Własności obiektu

### Atrybuty

- opisują określone **cechy** obiektu
- określają jednoznacznie rodzaj przechowywanej informacji (**typ danych**)
- są niepodzielne – reprezentują jedną wartość lub silnie związaną grupę wartości

### Usługi – operacje

- wskazują określone **zachowania** obiektu
- opisują **interfejs** obsługi obiektu
- implementacja (algorytm) usługi/operacji bywa nazywana **metoda**



## Usługi – interfejs obsługi obiektu

- dodatkowe informacje potrzebne dla wykonania usługi mogą zostać podane w postaci **parametrów**, zaś status jej wykonania może być udostępniany jako **rezultat**, który równocześnie może identyfikować stan obiektu
- liczba i typy parametrów oraz rezultatu (*protokół*) opisane są przez **sygnaturę**
- są wykonywane w odpowiedzi na zlecenia w postaci **komunikatów**, które powinny jednoznacznie identyfikować usługę oraz zawierać niezbędne parametry
- ta sama usługa może dotyczyć różnych obiektów (klas) gdzie może być inaczej implementowana (**polimorfizm**), jednak aby zapewnić możliwość zastępowania (podstawiania) musi mieć zawsze taką samą sygnaturę

## Klasy obiektów

- stanowią bazę większości obiektowo zorientowanych języków i metodologii, ale w niektórych podejściach ich rolę przejmują **obiekty prototypowe**
- różne znaczenia:
  1. wzorzec dla określonej kategorii strukturalnie podobnych obiektów (**typ obiektowy**)
  2. wzorzec j/w oraz mechanizm tworzenia obiektów według tego wzorca (**fabryka obiektów**)
  3. **zbiór obiektów** utworzonych według tego samego wzorca
- jeżeli klasy są jawnie wyróżniane w modelu/implementacji, to zakłada się, że każdy obiekt zna swoją klasę

## Klasy – przypadki szczególne

1. Klasie można przypisać własności wspólne dla wszystkich jej obiektów (**statyczne**).
2. Klasa może opisywać usługi/operacje **abstrakcyjne** (bez implementacji), ale wtedy nie może mieć bezpośrednich egzemplarzy (**klasa abstrakcyjna**).  
W skrajnym przypadku klasa może nie opisywać żadnej implementacji (atrybutów ani metod), tylko usługi/operacje abstrakcyjne – taka klasa bywa nazywana **interfejsem**.
3. Można przyjąć, że klasa jest również szczególnym obiektem – klasę takich obiektów nazywamy **metaklasą**.
4. **Klasa parametryczna** wymaga dodatkowych informacji (parametrów), aby stać się kompletnym wzorcem dla obiektów.

## Powiązanie

- w polskiej literaturze bywa używane w dwóch znaczeniach:  
**połączenie** (ang. *link*) – fizyczny lub logiczny związek między (najczęściej dwoma) obiektami  
**skojarzenie** (ang. *association*) – zbiór połączeń o takim samym znaczeniu i strukturze (np. klasach łączonych obiektów)  
połączenie jest egzemplarzem skojarzenia
- z natury jest dwukierunkowe, ale ze względów praktycznych bywa modelowane i realizowane jako jednokierunkowe
- dla więcej niż 2 obiektów (klas) jest trudne do modelowania i implementacji

## Powiązanie jako skojarzenie

- w uproszczeniu można rozumieć jako specyfikację atrybutów wzajemnie wskazujących połączone obiekty (należy jednak pamiętać, że powiązanie istnieje niejako ponad obiektami, nie w każdym z osobna)
- znaczenie opisuje się nazwą (często w określonym kierunku) lub rolą (rolami)
- ograniczenia mogą dotyczyć jednego lub wielu powiązań, najczęstszym jest krotność połączeń jednego obiektu
- niektóre powiązania wymagają dodatkowych informacji (w postaci atrybutów dla powiązania) lub też kwalifikacji – informacji wyróżniających podzbiór połączonych obiektów (w postaci atrybutów obiektu dla powiązania)

## Agregacja

- silna hierarchiczna forma powiązania z określoną semantyką: całość *składa się* z (wielu) części
- w wielu operacjach całość wraz z częściami traktowana jest jednostkowo (czego efektem może być propagacja operacji)
- może być rekursywna, części mogą przynależeć jednocześnie do różnych całości, mogą zmieniać swoją przynależność, mogą też istnieć niezależnie

## Kompozycja

- silna forma agregacji z czasem życia części ograniczonym do czasu życia całości: części mogą być tworzone równocześnie lub później niż całość oraz usuwane wcześniej lub razem z całością
- całość jest odpowiedzialna za zarządzanie tworzeniem i usuwaniem części
- część może przynależeć do dokładnie jednej całości
- w połączeniu z delegacją ról stanowi alternatywę dla wielokrotnego dziedziczenia

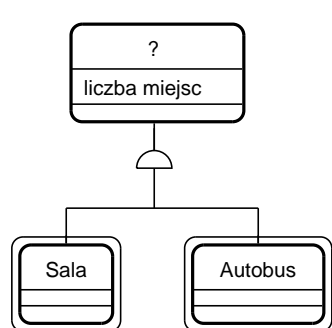
## Generalizacja

- wskazuje na podobieństwo między klasami (obiektami) w sensie posiadania przez specjalizację tych samych atrybutów, usług/operacji i relacji (realizowane w postaci **dziedziczenia**), a także tego samego znaczenia co generalizacja
- służy głównie **poszerzaniu** właściwości (specjalizacja może dodawać nowe atrybuty, usługi/operacje i relacje), ale także ich **modyfikacji**, a nawet **restrykcji** (specjalizacja może wprowadzać ograniczenia co do atrybutów i relacji oraz zmieniać implementacje usług/operacji, zachowując jednak ich semantykę oraz typy/sygnatury)

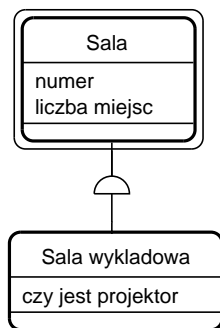
## Cechy generalizacji

- określa relację „bycia rodzajem” – egzemplarz (bezpośredni) specjalizacji jest także egzemplarzem (pośrednim) generalizacji – umożliwia podstawianie/zastępowanie
- tworzy struktury hierarchiczne, jest przechodnia i asymetryczna
- klasa-generalizacja może nie posiadać bezpośrednich egzemplarzy (**klasa abstrakcyjna**)

## Typowe błędy generalizacji

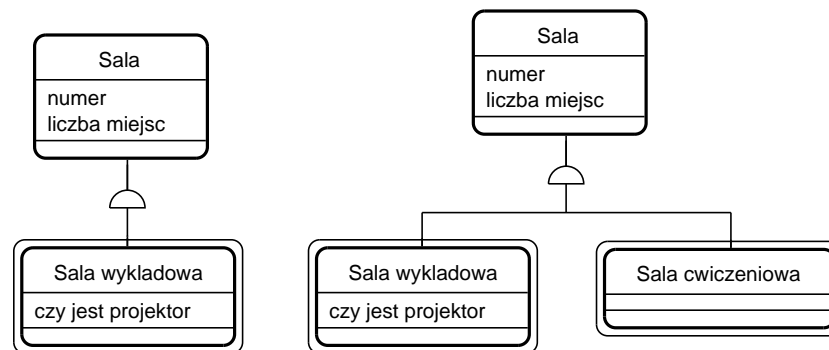


brak uzasadnienia  
w dziedzinie



klasa abstrakcyjna  
na dole struktury

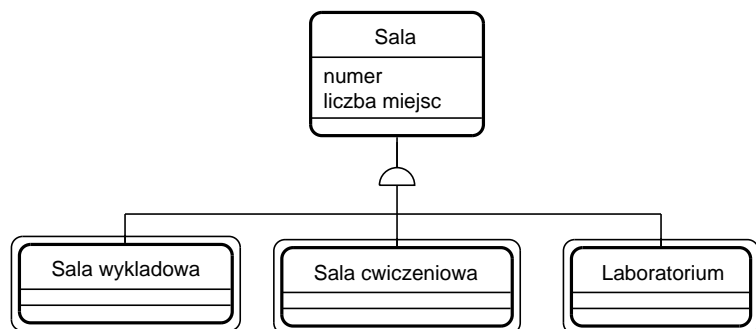
## Typowe błędy generalizacji – cd.



pojedyncza  
klasa-generalizacja

brak specyficznych  
atrybutów/usług/relacji

## Typowe błędy generalizacji – cd.

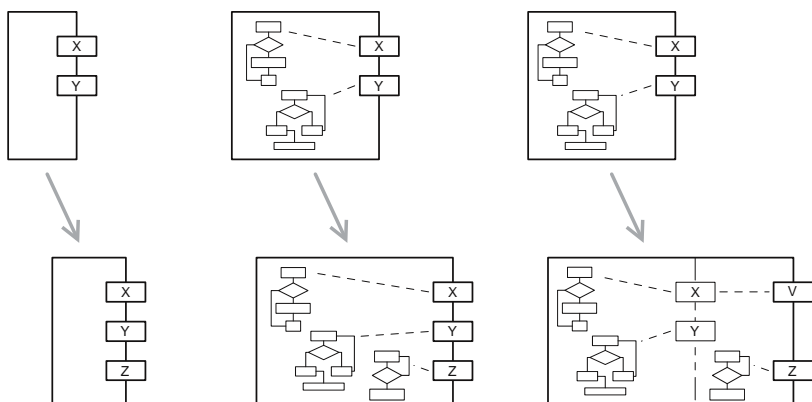


rozdzielanie obiektów przez przynależność do klasy

## Dziedziczenie

- jeden z najważniejszych, a zarazem najbardziej kłopotliwych elementów podejścia obiektowego
  - różne aspekty:
    1. wykorzystywanie struktury wewnętrznej i zdefiniowanych zachowań – dziedziczenie implementacji (ang. *subclassing*) – pozwala na ich wielokrotne użycie
    2. zgodność na poziomie dostępnych usług – dziedziczenie interfejsów (ang. *subtyping*) – realizuje generalizację
    3. możliwość podstawiania/zastępowania (ang. *substitutability*)
- (1 może służyć 2 może służyć 3)

## Dziedziczenie interfejsu i implementacji

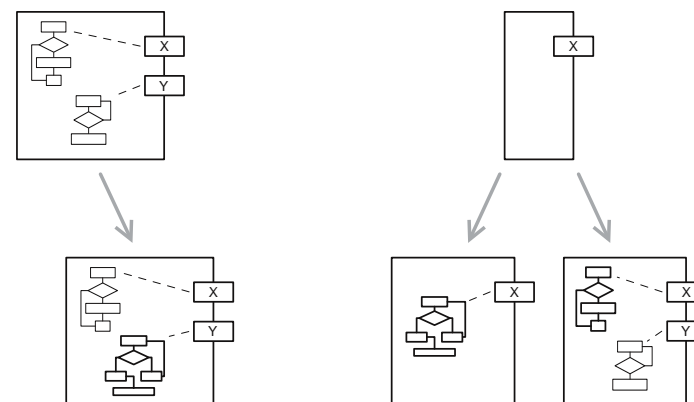


dziedziczenie interfejsu

dziedziczenie interfejsu oraz implementacji

dziedziczenie samej implementacji

## Polimorfizm a dziedziczenie

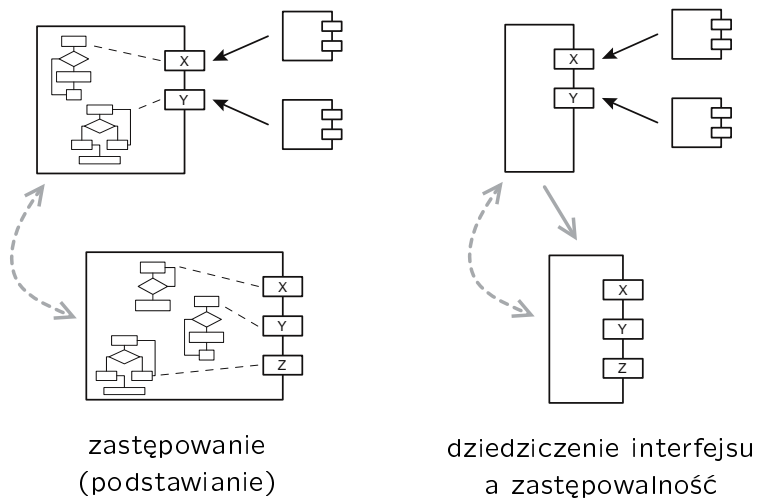


polimorfizm przy bezpośrednim dziedziczeniu

polimorfizm przy dziedziczeniu ze wspólnej generalizacji

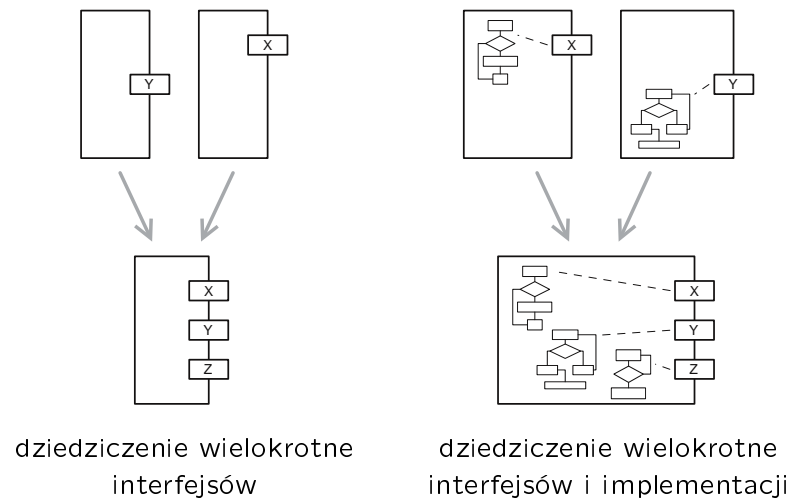


## Zastępowalność



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Dziedziczenie wielokrotne



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## wielokrotne dziedziczenie implementacji

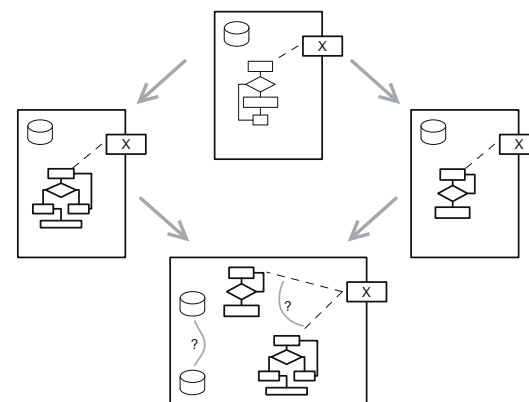
**zaleta:** uwytłumnia specjalizację pozwalając na jawną specyfikację wspólnych cech

**wada:** struktury kratowe (dziedziczenie z różnych klas o wspólnej nadklasie) zwiększają złożoność modelu i utrudniają implementację

**problem:** konflikt stanu i zachowań dziedziczonych po wspólnej nadklasie

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Problem kraty w dziedziczeniu



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Unikanie wielokrotnego dziedziczenia

- zagnieżdżone dziedziczenie

**wada:** wielokrotna specyfikacja interfejsów i implementacji, duże hierarchie dziedziczenia

- wielokrotne dziedziczenie interfejsów

**wada:** wielokrotna specyfikacja implementacji

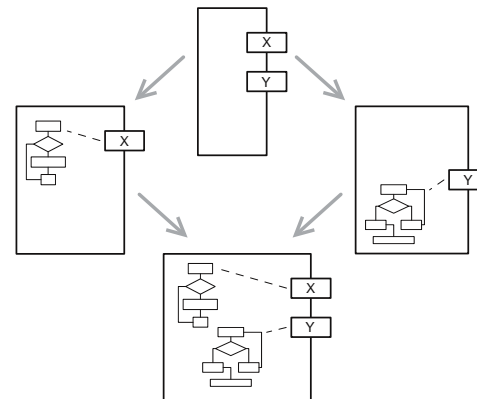
- agregacja (kompozycja) implementacji z delegacją roli (przekazywaniem wykonania usługi do innego obiektu)

**wada:** utrata identyfikacji obiektu

- **kompromis:** dziedziczenie najważniejszej implementacji i wielokrotne dziedziczenie interfejsów oraz kompozycja implementacji z delegacją ról

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Wielokrotne dziedziczenie fragmentów implementacji interfejsu (ang. *mixin*)



Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Hermetyzacja a ukrywanie informacji

- wyróżnić można wiele kategorii interfejsów, np.:

**publiczny** – dostępny dla wszystkich innych składowych

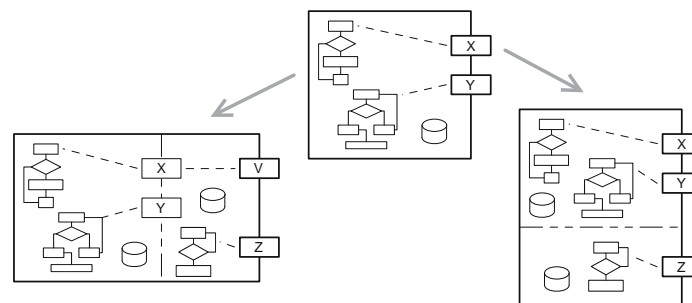
**dziedziczenia** – dostępny dla specjalizacji

- interfejsy powinny odkrywać możliwie mało z wewnętrznej struktury składowej (możliwie niezależne składowe), co ułatwia budowę systemu/modelu i wprowadzanie zmian

- poszczególne interfejsy powinny być specyfikowane w kategoriach dostępnych (**eksportowanych**) usług; stan (atrybuty) powinien być udostępniany jedynie poprzez usługi (eksportowanie atrybutu równoważne jest udostępnieniu usług odczytywania i zapisywania jego wartości)

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## Interfejsy dziedziczenia



**abstrakcje typu białej skrzynki** – implementacja całkowicie udostępniana specjalizacjom

**abstrakcje typu czarnej skrzynki** – implementacja całkowicie zakryta za interfejsem – ułatwia wielokrotne użycie

Copyright (c) 1997–2001 Marek Kisiel-Dorohinicki

## techniki wielokrotnego użycia

na poziomie pojedynczego modelu/systemu:

- wykorzystanie klas – wzorców dla grup obiektów
- delegacja ról (często w połączeniu z kompozycją) – przekazywanie zlecenia wykonania usługi do innej usługi/obiektu
- dziedziczenie implementacji

między modelami/systemami:

- biblioteki klas do bezpośredniego wykorzystania lub specjalizacji
- wzorce projektowe