

Programowanie obiektowe

Grzegorz Jabłoński

Katedra Mikroelektroniki i Technik
Informatycznych (K22)

Budynek B18

gwj@dmcs.p.lodz.pl

(631) 26-48

Program przedmiotu

<http://neo.dmcsp.lodz.pl/po>

Ogólne spojrzenie na język C++

Klasy

Pola i metody

Przeciążenie operatora

Dziedziczenie

Funkcje wirtualne

Wzorce

Obsługa wyjątków

Hierarchie klas

Biblioteka standardowa C++ (STL)

Dzisiejszy wykład

- # Cele projektowania
- # Paradygmaty programowania
- # Proces projektowania obiektowego
- # Podstawy projektowania obiektowego
 - # Abstrakcja
 - # Interfejsy
 - # Zadania
 - # Współpracownicy
- # Przykład
 - # Identyfikacja obiektów
 - # Identyfikacja relacji

Cele projektowania

Ponowne użycie

- # Opracowanie przenośnych i niezależnych komponentów, które mogą być ponownie użyte w wielu systemach

Rozszerzalność

- # Wsparcie dla zewnętrznych modułów rozszerzających (np. Photoshop plug-ins)

Elastyczność

- # Łatwość zmian przy dodaniu dodatkowych danych/możliwości
- # Małe prawdopodobieństwo totalnego uszkodzenia systemu przy zmianach w projekcie
- # Lokalne efekty zmian

Proces projektowania

- # Cel: zbudować system
- # Proces projektowania przebiega następująco:
 - # Podział/opis systemu jako zespołu komponentów
 - # Podział/opis komponentów jako zespołu mniejszych komponentów
- # Pojęcie abstrakcji
 - # Podstawowe w procesie projektowania, ukrywa szczegóły komponentów nieistotne w bieżącej fazie projektowania
- # Identyfikacja komponentów metodą zstępującą
 - # Stopniowy podział systemu na coraz mniejsze, prostsze komponenty
- # Integracja komponentów metodą wstępującą
 - # Budowa systemu przez składanie komponentów na różne sposoby
- # Projekt odbywa się zgodnie z paradigmatem: proceduralnym, modularnym, obiektowym

Abstrakcja

- # Nazwany zbiór atrybutów i sposobu zachowania konieczny do modelowania obiektu w określonym celu
- # Pożądane właściwości
 - # Dobra nazwa nazwa oddaje cechy abstrakcji
 - # Spójny sensowny
 - # Dokładny zawiera tylko atrybuty modelowanego obiektu
 - # Minimalny zawiera tylko atrybuty niezbędne dla określonego celu
 - # Kompletny zawiera wszystkie atrybuty niezbędne dla określonego celu

Formy abstrakcji

Funkcje (projektowanie proceduralne)

- # Zdefiniowanie zbioru funkcji w celu wykonania zadania
- # Przekazywanie informacji między funkcjami
- # Wynik: hierarchiczna organizacja funkcji

Moduły (projektowanie modularne)

- # Zdefiniowanie modułów, w których są dane i procedury
- # Każdy moduł posiada sekcję prywatną i publiczną
- # Moduł grupuje powiązane dane i procedury
- # Moduł działa jako mechanizm zasięgu

Klasy/obiekty (projektowanie obiektowe)

- # Abstrakcyjne typy danych
- # Podział projektu na zbiór współpracujących klas
- # Każda klasa pełni bardzo szczególne funkcje
- # Klasy mogą być użyte do tworzenie wielu egzemplarzy obiektów

Paradygmat proceduralny

- # Zastosowanie dekompozycji proceduralnej
 - # Podział problemu na sekwencję prostszych podproblemów rozwiązywanych niezależnie
- # Program składa się z sekwencji wywołań procedur
- # Projektant myśli używając pojęć zadań i podzadań, identyfikując co musi być zrobione na jakich danych
- # Tradycyjne języki proceduralne: COBOL, FORTRAN, C
- # Notacja projektowa: diagramy strukturalne, diagramy przepływu danych

Problemy podejścia proceduralnego

- # Otrzymujemy duży program złożony z wielu małych procedur
- # Brak naturalnej hierarchii organizującej te procedury
- # Często nie jest jasne, która procedura co wykonuje na których danych
- # Słaba kontrola potencjalnego dostępu procedur do danych
- # Powyższe cechy powodują, że usuwanie błędów, modyfikacja i pielęgnacja są trudne
- # Naturalna wzajemna zależność procedur spowodowana przekazywaniem danych (albo, co gorsza, danymi globalnymi) powoduje, że trudno jest je ponownie użyć w innych systemach

Przykład podejścia proceduralnego

- # Rozważmy dziedzinę zastosowań geometrycznych (kształty, kąty, dodawanie punktów i wektorów)

```
void distance(int x1, int y1, int x2, int y2, float& distance);  
void angle2radian(float degree, float& radian);  
void radian2angle(float radian, float& degree);  
void circlearea(int centerx, int centery, int radius, float& area);  
void squarearea(int x1, int x2, int width, int height, float &area);  
void squareperimeter(int x1, int x2, int width, int height, float &prm);  
...
```

- # Centralnym aspektem projektu jest procedura, nie dane
 - # W rzeczywistości brak oddzielnej reprezentacji danych

Programowanie modularne

- # Względnie proste rozszerzenie czystego podejścia proceduralnego
- # Dane i związane z nimi procedury są zebrane w modułach
- # Moduł zapewnia jakąś metodę ukrycia jego zawartości
- # W szczególności, dane mogą być modyfikowane tylko przez procedury w tym samym module
- # Proces projektowania uwypukla dane w stosunku do procedur. Najpierw identyfikujemy niezbędne elementy danych, a potem dopisujemy procedury, które na nich operują
- # Typowe języki programowania: Ada 83, Modula

Problemy w projektowaniu modularnym

- # Moduły rozwiązują większość problemów z programowaniem proceduralnym wymienionych wcześniej.
- # Moduły pozwalają na jedynie częściowe ukrywanie informacji w porównaniu z podejściem obiektowym
- # Nie można mieć kilku kopii jednego modułu, co ogranicza projektanta

Przykład projektowania modułarnego

```
// Geometry Module

struct Circle { int centerx, centery; int radius; };

struct Square { int x1, x2, width, height; };

Circle *NewCircle(int center, int radius);

Square *NewSquare(int x1, int x2, int width, int height);

float CircleArea(Circle& c);

float SquareArea(Square& s);

float SquarePerimeter(Square& s);

void distance(int x1, int y2, int x2, int y2, float& distance);

void angle2radian(float degree, float& radian);

void radian2angle(float radian, float& degree);

...
```

- # Centralnym aspektem projektu jest procedura, ale występuje również reprezentacja danych
- # Pojęcie punktu nie wprowadzone, bo nie jest potrzebne w projekcie i nie byłoby z tego żadnych korzyści

Paradygmat obiektowy

- # Analogia do konstruowania maszyny z części składowych
- # Każda część jest obiektem, który posiada swoje atrybuty i właściwości oraz współdziała z innymi częściami w celu rozwiązania problemu
- # Identyfikacja klas obiektów, które mogą być ponownie użyte
- # Myślenie używając pojęć obiektów i ich wzajemnego oddziaływania
- # Na wysokim poziomie abstrakcji, myślenie o obiektach jako bytach samych w sobie, nie wewnętrznych strukturach potrzebnych do działania obiektu
- # Typowe języki obiektowe: Smalltalk, C++, Java, Eiffel

Dlaczego podejście obiektowe?

- # To po prostu kolejny paradygmat ... (i zapewne będą kolejne)
- # Każdy system zaprojektowany i zaimplementowany obiektywnie może być zbudowany używając czystego podejścia proceduralnego
- # Podejście obiektowe jednakże ułatwia pewne rzeczy
- # Podczas projektowania na wysokim poziomie, często bardziej naturalne jest myślenie o problemie używając pojęć zespołu oddziałujących na siebie rzeczy (obiektów), niż pojęć danych i procedur
- # Podejście obiektowe często ułatwia zrozumienie i kontrolę nad dostępem do danych
- # Podejście obiektowe promuje ponowne użycie

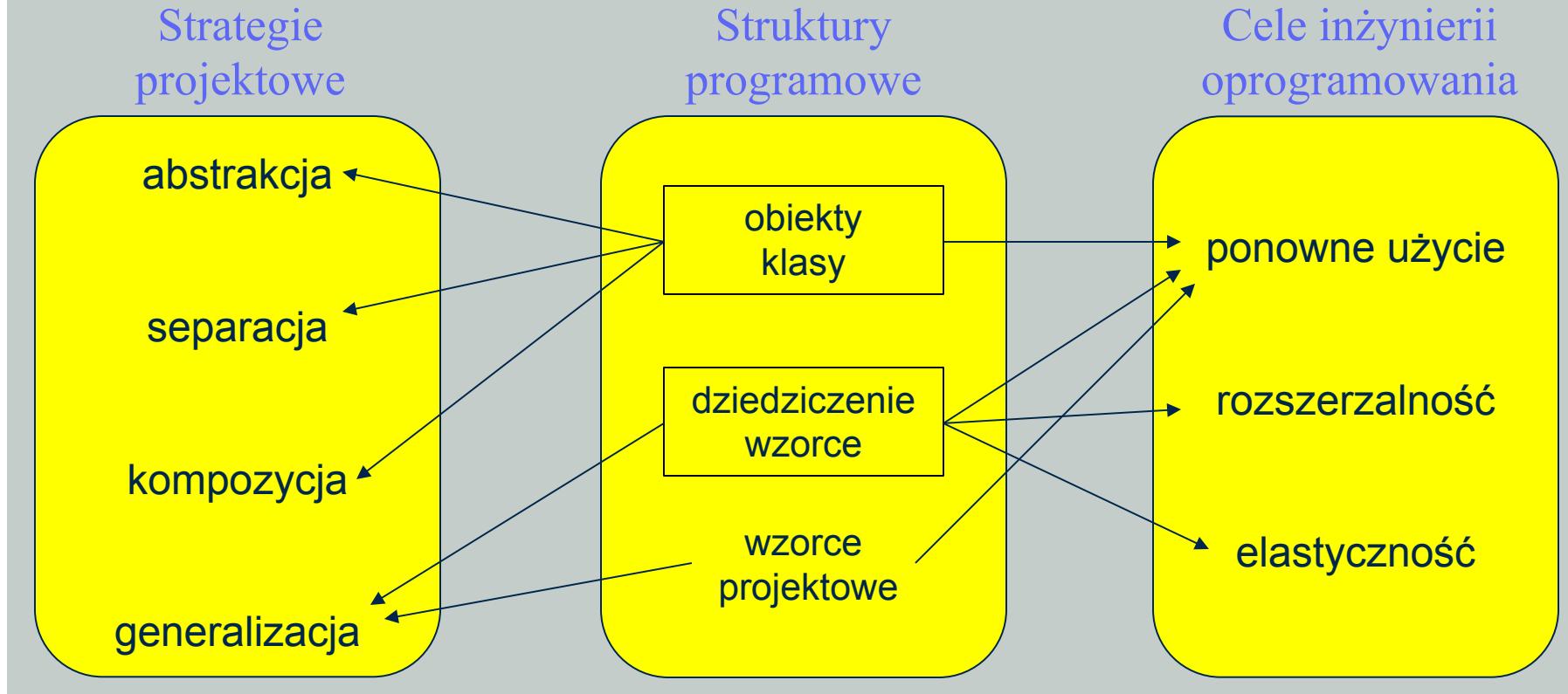
Przykład projektu obiektowego

```
class Point { ...  
    float distance(Point &pt);  
};  
  
class Shape { float Area(); float Perimeter(); Point center(); }  
  
class Circle : Shape {  
private: Point center; int radius;  
public: // constructors, assignment operators, etc...  
    float Area(); // calc my area  
    float Perimeter();  
};  
  
class Square : Shape {  
private: Point anchor; int width, height;  
public: // constructors, assignment operators, etc...  
    float Area();  
    float Perimeter();  
};  
...
```

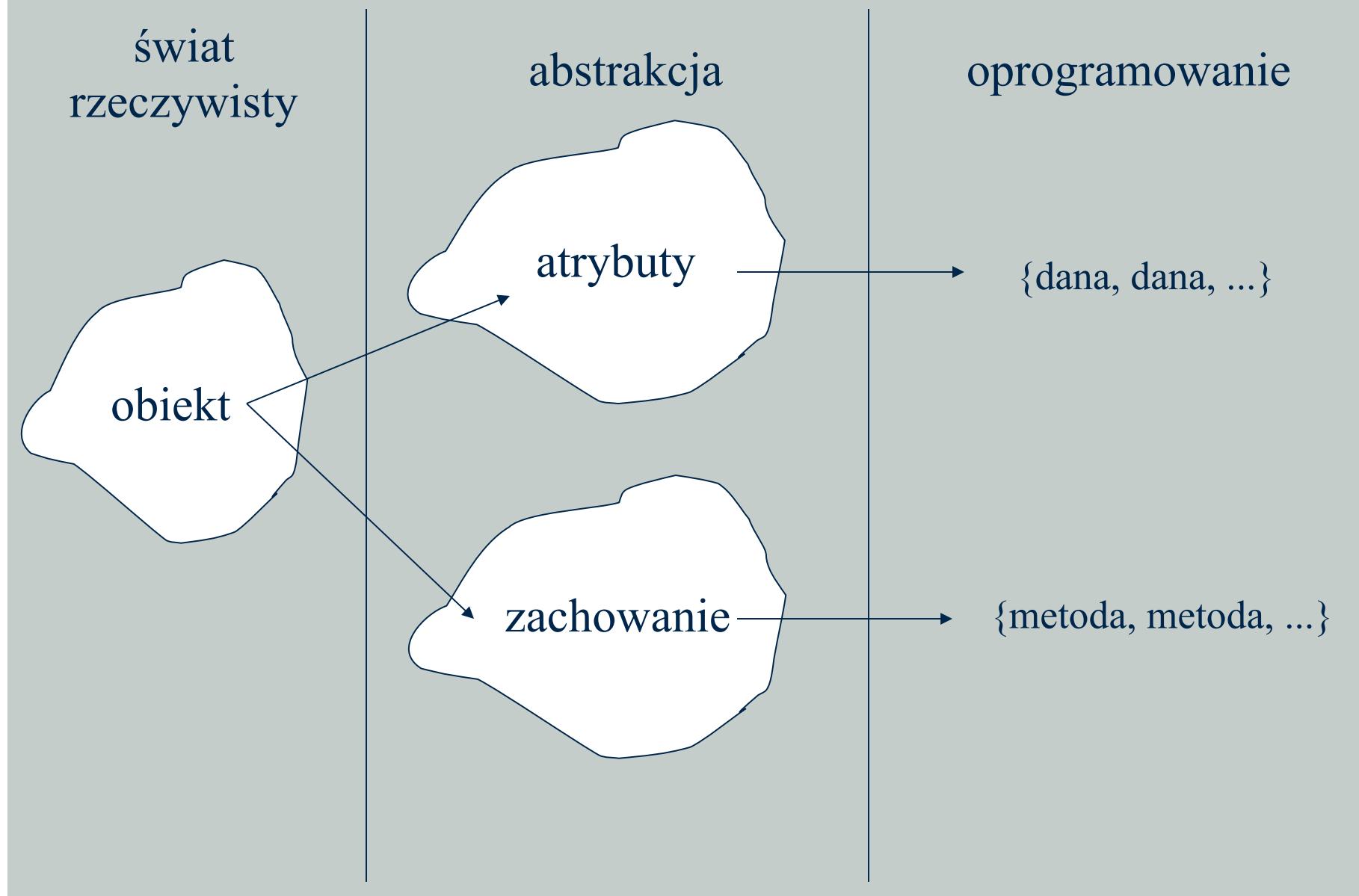
- # Centralnym aspektem projektu są teraz dane, operacje są zdefiniowane razem z danymi

Strategie projektowe w podejściu obiektowym

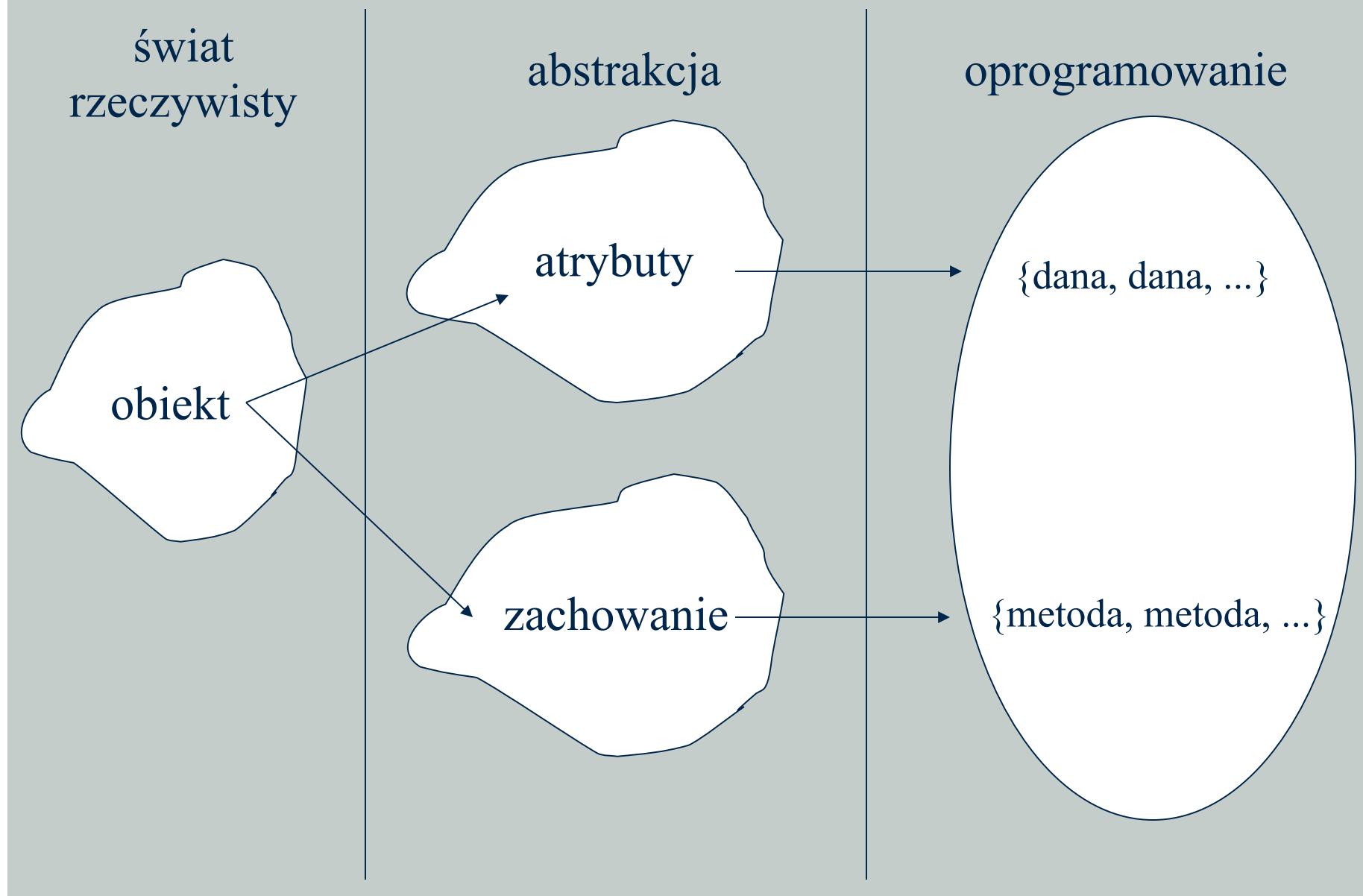
- # Abstrakcja modelowanie niezbędnych właściwości
- # Separacja oddzielenie „co” od „jak”
- # Kompozycja budowanie złożonych struktur z prostszych
- # Generalizacja identyfikacja elementów wspólnych



Odwzorowanie abstrakcji i oprogramowania

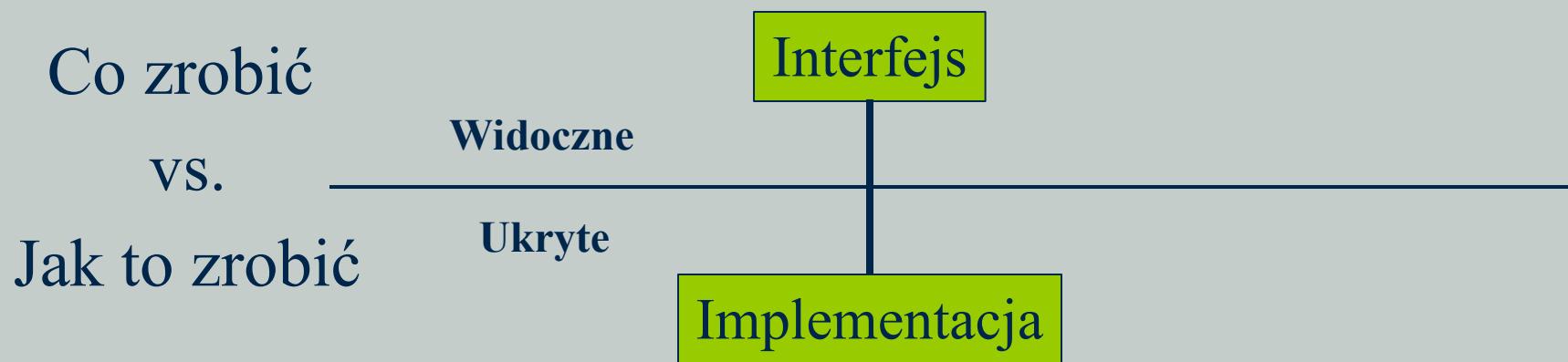


Odwzorowanie abstrakcji i oprogramowania (OO)



Oddzielenie interfejsu od implementacji

- # W programowaniu: niezależna specyfikacja interfejsu i jednej lub wielu implementacji tego interfejsu.

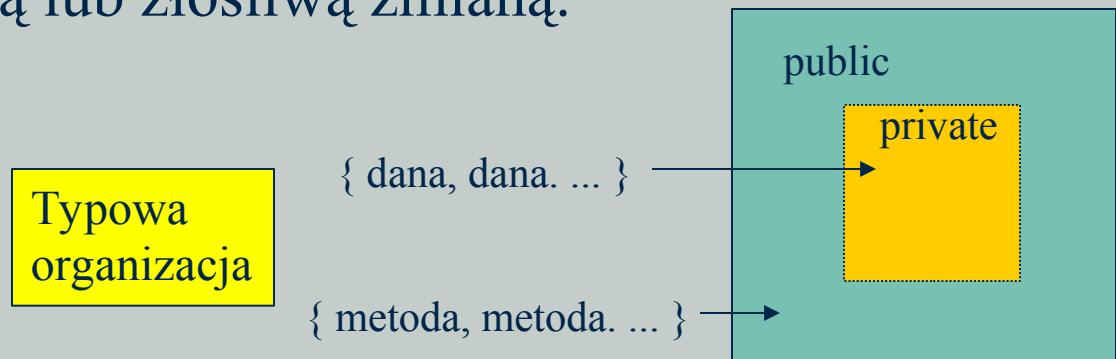


- # Dodatkową korzyścią jest możliwość programowania uzależnionego od interfejsu bez zajmowania się jego implementacją
 - # Programowanie oparte na kontraktcie
 - # Umożliwia abstrakcję w procesie projektowania

Ogólna struktura klasy

Klasa

- # nazwana reprezentacja programowa abstrakcji która oddziela implementację reprezentacji od interfejsu reprezentacji
- # Klasa modeluje abstrakcję, która modeluje obiekt (być może "rzeczywisty")
- # Klasa reprezentuje wszystkich członków grupy obiektów ("egzemplarze" klasy)
- # Klasa dostarcza publiczny interfejs i prywatną implementację
- # Ukrywanie danych i algorytmów przed użytkownikiem jest ważne. Ograniczenia dostępu zabezpieczają (częściowo) przed przypadkową, błędna lub złośliwą zmianą.



Proces projektowania obiektowego

- # Ograniczenie obszaru zastosowań: przypadki użycia (scenariusze, opisy użytkowania)
 - # Identyfikacja **obiektów** (dane)
 - # Identyfikacja zadań (zachowanie)
- # Definicja zachowania
 - # Identyfikacja współpracowników
 - # Czy zachowanie jest osiągnięte przez pojedynczą klasę, czy przez współpracę "spokrewnionych" klas
 - # Zachowanie statyczne
 - Zawsze się tak samo zachowuje
 - # Zachowanie dynamiczne
 - W zależności od warunków (rodzaj, źródło wywołania) zachowanie jest możliwe lub nie
 - # Identyfikacja relacji między obiektami
 - # Kompozycja przez asocjację, agregację, inne

Początek

Na początku... jest specyfikacja

Specyfikacja:

Zaprojektować katalog płyt z muzyką. System musi umożliwiać dodawanie płyt, przechowywanie informacji o wykonawcach, tytułach albumów, tytułach utworów, kompozytorach itp. Użytkownik systemu powinien mieć możliwość wyszukiwania dowolnej informacji w kolekcji. Powinien także umożliwiać przeglądanie kolekcji przez użytkownika.

- # Specyfikacja jest zwykle niewystarczająca
- # Wiele istotnych rzeczy nie jest powiedziane
- # Często zawiera wiele stwierdzeń nieistotnych

Identyfikacja obiektów

Należy

- # zidentyfikować potencjalne obiekty w specyfikacji
- # wyeliminować fałszywych kandydatów
- # określić interakcje między "prawdziwymi" obiektami
- # stworzyć klasy z obiektów

Ten proces:

- # wymaga doświadczenia żeby go prawidłowo przeprowadzić
- # istnieją standardowe podejścia do problemu, nie zawsze w pełni dostosowane do konkretnej sytuacji
- # często kilka podejść jest używanych jednocześnie
- # powinien raczej prowadzić do zbyt dużej, a nie zbyt małej liczby obiektów

Kilka podejść do problemu

Abbott and Booch

- # używać rzeczowników i zaimków do zidentyfikowania obiektów i klas
- # liczba pojedyncza -> obiekt, liczba mnoga -> klasa
- # nie wszystkie rzeczowniki zostaną obiektemi

Coad and Yourdon:

- # identyfikować pojedyncze lub grupowe "rzeczy" w systemie/problemie

Ross sugeruje kilka powszechnych kategorii obiektów

- # ludzie
- # miejsca
- # rzeczy
- # organizacje
- # pojęcia
- # wydarzenia

Obiekty i dziedzina problemu

- # Co jest "potencjalnym obiektem" zależy od dziedziny problemu
- # Należy dyskutować z ekspertem w danej dziedzinie - osobą, która pracuje w dziedzinie, w której system będzie pracował
- # Należy starać się zidentyfikować obiekty na podstawie sposobu myślenia eksperta o problemie

Specyfikacja:

Zaprojektować katalog płyt z muzyką. System musi umożliwiać dodawanie płyt, przechowywanie informacji o wykonawcach, tytułach albumów, tytułach utworów, kompozytorach itp. Użytkownik systemu powinien mieć możliwość wyszukiwania dowolnej informacji w kolekcji. Powinien także umożliwiać przeglądanie kolekcji przez użytkownika.

Eliminacja "fałszywych" obiektów

Obiekt powinien:

- # być bytem występującym w świecie rzeczywistym
- # być istotnym elementem wymagań
- # mieć ściśle określona granicę
- # mieć sens - atrybuty i zachowanie powinny być powiązane

Złe znaki:

- # nazwa klasy jest czasownikiem
- # klasa jest opisana jako wykonywanie operacji
- # klasa obejmuje wiele abstrakcji
- # klasa ma tylko jedną metodę publiczną
- # klasa nie ma metod

Przykład: katalog płyt

- # Wyszukiwanie rzeczowników
- # Pierwsza próba:
 - # muzyka
 - # katalog (kolekcja)
 - # system
 - # użytkownik
 - # utwór
 - # tytuł
 - # wykonawca
 - # album (płyta)
 - # kompozytor
 - # informacja

Specyfikacja:

Zaprojektować **katalog płyt z muzyką**. System musi umożliwiać dodawanie płyt, przechowywanie informacji o **wykonawcach, tytułach albumów, tytułach utworów, kompozytorach** itp. **Użytkownik systemu** powinien mieć możliwość wyszukiwania dowolnej **informacji** w **kolekcji**. Powinien także umożliwiać przeglądanie **kolekcji** przez **użytkownika**.

Przykład: katalog płyt

Odrzucamy (na razie):

- # muzyka (odnosi się do rodzaju przechowywanej informacji, ale nie przechowujemy muzyki)
- # katalog (kolekcja, system) - znaczy to samo
- # informacja - ogólna nazwa elementów kolekcji
- # użytkownik - zewnętrzny w stosunku do systemu, gra rolę w systemie
- # utwór, tytuł, wykonawca, kompozytor

Wymagane struktury danych:

- # katalog zawiera kolekcję nagrań
- # album ma tytuł, wykonawcę, listę utworów
- # utwór ma tytuł, kompozytora, wykonawcę

Czy tytuł jest klasą?
A nazwisko?

Przykład: katalog płyt

Co z ogólnym sterowaniem całością?

- # Główny sterownik może być procedurą lub obiektem
- # Kolekcja
 - # Użytkownik używa katalogu, który zawiera kolekcję, listę obiektów typu płyta.
 - # Umożliwia wykonanie każdej z żądanych operacji
- # Kolekcja powinna reagować na zdarzenia, a nie aktywnie ich poszukiwać. W implementacji konieczne jest przetwarzanie pliku wejściowego lub graficzny interfejs użytkownika (GUI). To nie jest część kolekcji, aczkolwiek będzie z nią współpracować.

Ogólna struktura obiektu

- # Obiekt:
 - # osobny egzemplarz pewnej klasy który ukrywa szczegóły implementacji i jest strukturalnie identyczny z wszystkimi obiektami danej klasy
- # Obiekt łączy dane i operacje, które mogą być wykonywane na tych danych
 - # składowe = pola + metody
- # Dane prywatne obiektu są dostępne **jedynie** za pośrednictwem metod klasy
- # Obiekt ukrywa szczegóły implementacyjne przed użytkownikiem

