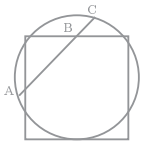
Metodologie obiektowe

Radosław Klimek

2001-9



http://home.agh.edu.pl/rklimek

Wprowadzenie do podejścia obiektowego

Percepcja i klasyfikowanie

Trzy podstawowe metody percepcji i klasyfikacji mające wpływ na myślenie człowieka:

1. różnicowanie pomiędzy poszczególne obiekty i ich atrybuty – przykładowo różnicowanie pomiędzy drzewem i jego rozmiarami w relacji przestrzennej do innych oiektów;
2. rozróżnienie pomiędzy całymi obiektami a ich częściami składowymi – przykładowo drzewo składające się z gałęzi czy pnia, które to części składają się na całe drzewo;
3. konstruowanie klas obiektów i rozróżnianie ich – przykładowo klasa wszystkich drzew oraz klasa wszystkich kamieni.

Człowiek ma naturalne skłonności do klasyfikowania.

Tu już jest używane pojęcie obiektu, ale jest ono używane bardzo ogólnie, w zasadzie można by mówić np. o elementach.

Zarządzanie złożonością systemów informatycznych

Istnieją pewne zasady zarządzania złożonością współczesnych i coraz bardziej skomplikowanych systemów informatycznych:

— abstrakcja – ignorowanie od nieistotnych aspektów z punktu widzenia bieżącego celu. Istnieje abstrakcja proceduralna i abstrakcja danych.

— hermetyzacja, lub enkapsulacja – ukrywanie informacji w pewnym miejscu, z jednoczesnym zamknięciem pojedynczej decyzji projektowej w tym miejscu. Abstrakcja może być uznana jako jedna z form hermetyzacji.

— dziedziczenie – pewne podobieństwo wynikające z wykorzystania niektórych cech jednych elementów przez elementy inne. Także, inaczej, specjalizacja-generalizacja.

— skojarzenie – wskazywanie związków pomiędzy pewnymi elementami, np. samochód i właściciel.

Zarządzanie złożonością systemów informatycznych (cd.)

— porozumiewanie się poprzez przekazywanie komunikatów – to zostawianie informacji przez pewne byty, w zazwyczaj ustalonych miejscach, do odbioru przez inne byty.

— rozpoznanie i ujednolicenie metod klasyfikacji – już wspominane,

— problem skali – to zasada całość-część, to notacje i strategie prowadzące, w połączeniu ze skalą, poprzez większy model/do większego modelu.

— kategorie zachowań – klasyfikowanie dynamiki systemu: (1) poprzez związki przyczynowe, (2) poprzez podobieństwo ewolucji, (3) ze względu na podobieństwo funkcji.

Inżynieria oprogramowania

W wielu krajach wytwarzanie oprogramowania ma duży wpływ na PKB, wyprzedzając inne dziedziny, takie np. jak: przemysł wytwórczy, sektor bankowo-ubezpieczeniowy, a nawet przemysł energetyczny! – szacuje się że w kosztach opracowania telefonii komórkowej koszty oprogramowania wynoszą ok. 75%.

O wzroście złożoności oprogramowania, wzrastającym zapotrzebowaniu na nie i coraz większych wymaganiach świadczą następujące dane:

— w USA jedynie 2% systemów jest używanych w wersji dostarczonej pierwotnie, 29% nie zostało nigdy dostarczonych do klienta, mimo że zostały zapłacone, a 47% nie jest nigdy wykorzystywane, mimo że zostały dostarczone, z powodu niedostosowania do potrzeb;

— w 1990 r. ważne programy bywały dostarczane z jednorocznym opóźnieniem i tylko 1% dużych projektów programistycznych zostało ukończonych na czas i w granicach projektowanego budżetu, informatyka jest bez wątpienia jedną z ostatnich gałęzi w której można aktualnie obserwować odchylenia większe niż 50% pomiędzy terminami i/lub kosztami planowania na początku projektu i stwierdzonymi po jego zakończeniu;

— udział kosztów obsługi w 1995 r. był szacowany na ponad 70% obrotu dla firm pracujących w dziale informatyki, a 22% tych kosztów jest związane z poprawianiem oprogramowania.

Podejście strukturalne a obiektowe

Podejście strukturalne
Rozpatrywanie zbioru procedur i funkcji działających na pewnych strukturach danych – może to okazać się niewystarczające z punktu widzenia rozpatrywanego i złożonego świata rzeczywistego.
Podejście obiektowe
W podejściu tym istotna jest spójność modeli opisujących oprogramowanie, uzyskiwana poprzez zdefiniowanie i zamknięcie w pojedynczej jednostce zarówno danych (struktury) jak i operacji z nimi związanych – tą jedną jednostką jest właśnie *obiekt*, pomyślany jako podstawowy i elementarny komponent.

Podejście obiektowe – pierwsze podsumowanie

Koncepcje podejścia obiektowego mają pozytywny wpływ na proces modelowania systemów. Wynika to faktu, że:

1. obiektowość jest pewną ideą, która ma przybliżyć świat informatyczny do świata rzeczywistego, poprzez:
   (a) ogólne podwyższenie poziomu abstrakcji w procesie analizy, projektowania i programowania,
   (b) dopasowanie technologii informatycznych inżynierii oprogramowania do oczekiwań analityków i projektantów – celem podejścia obiektowego jest uzyskanie jak najmniejszej luki pomiędzy myśleniem o rzeczywistości, a myśleniem o procesach zachodzących w systemie informatycznym;
2. podejście obiektowe umożliwia przedstawienie systemu z zastosowaniem tego samego modelu, począwszy od fazy początkowej (analiza) aż do końcowej (zazwyczaj implementacja) – model obiektowy przez wszystkie fazy projektu jest co prawda rozwijany i podlega licznym modyfikacjom, ale też jest to ten sam model, tyle że coraz bardziej „dojrzały".

Podejście obiektowe – pierwsze podsumowanie (cd.)

Można także powiedzieć, że celem obiektowości jest niegubienie semantyki danych i przybliżeniejej do świata rzeczywistego.

Model w podejściu obiektowym przez wszystkie fazy projektu był rozwijany, ale też był to ten sam model, tyle że coraz bardziej „dojrzały" co pozwala uniknąć pewnego rozejścia się struktur danych i operacji, tak charakterystycznego, jako niezamierzony efekt uboczny, dla podejścia strukturalnego.

Wpływ podejścia obiektowego na różne obszary informatyki

Podejście obiektowe oddziaływuje na wiele obszarów współczesnej informatyki:

- metody analizy i projektowania systemów informatycznych (np. takie pojęcia jak: obiekty, klasy, metody, dziedziczenie, hermetyzacja, polimorfizm);
- języki programowania (np. Smalltalk, C++, Eiffel, Ada95);
- bazy danych i składy obiektów trwałych (standardy: ODMG, Gemstone i inne),
- współdziałanie systemów heterogenicznych (np.: CORBA),
- inne (wizyjne środki programistyczne, biblioteki oprogramowania oraz inne).

Metodyki i metodologie

DEFINICJA 1. Przez *metodykę* rozumiemy zestaw **pojęć**, **notacji**, **modeli**, **języków**, **technik i sposobów postępowania** służących do analizy dziedziny stanowiącej przedmiot projektowanego systemu oraz do projektowania pojęciowego, logicznego i/lub fizycznego. Z każdą metodyką jest związana *notacja*. ⌟

Metodyka ustala:

- fazy projektu,
- modele tworzone w każdej z faz,
- scenariusze postępowania w każdej z faz,
- reguły przechodzenia od fazy danej do następnej,
- notacje które należy używać,
- dokumentację powstającą w każdej z faz.

Metodyka dyscyplinuje przebieg procesu analizy i projektowania, pozwalając tym samym na w miarę obiektywne rozliczenie (także czasowe i finansowe) jego uczestników.

Metodyki obiektowe

DEFINICJA 2. *Metodyka obiektowa* (ang. *object-oriented methodology*) to metodyka wykorzystująca pojęcia obiektowości dla celów modelowania pojęciowego oraz analizy i projektowania systemów informatycznych. Podstawowym składnikiem tych metodyk jest diagram obiektów, będący zwykle wariantem notacyjnym i pewnym rozszerzeniem diagramów encja-związek, a także szereg innych diagramów. ⌟

- Obserwowana była (eksplozja) metodyk i notacji obiektowych, których obecnie jest kilkadziesiąt, np.: BON, Catalysis, DOOS (Wirfs/Brock), EROOS, Express, Fusion, Goldberg/Rubin, MainstreamObjects, Martin/Odell, MOSES, Objectory (Jacobson), OMT (Rumbaugh), OOA/OOD (Coad/Yourdon), OODA (Booch), OSA, Sintropy, OOSA (Shlaer/Mellor), UML oraz inne.
- Zauważa się już jednak koncentrację i zredukowanie metodyk (UML, OPEN , BON i inne), a na czoło wybija się ostatnio notacja UML autorstwa znanych metodologów Jacobsona, Rumbaugha i Boocha.

Trochę historii podejścia obiektowego

Podejście obiektowe pojawiło się w późnych latach sześćdziesiątych przy okazji języka *Simula 67**, który jest uważany za prekursora (przodka) obiektowości w programowaniu. Sam język powstał w Norweskim Ośrodku Obliczeniowym** w Oslo w 1967 r. Jego twórcami byli: Ole-Johan Dahl, Bjørn Myhrhauga i Kristen Nygaard.

Język Simula był językiem uniwersalnym wysokiego poziomu z wbudowanymi mechanizmami do symulacji. Simula miała udogodnienia dla obliczeń numerycznych, działań na tekstach, działań na zbiorach i strukturach listowych, a także operacje we/wy.

Z punktu widzenia obiektowości, Simula zawierała klasy, podklasy, wirtualne funkcje i aktywne obiekty.

Innym znanym przykładem języka obiektowego jest *Smalltalk*, opracowany w latach 1976-83 w Xerox Palo Alto Research Center w Kalifornii. Język wprowadza klasy, podklasy, wirtualne funkcje, przesyłanie komunikatów, metaklasy. Wszystko jest tu obiektem, w szczególności liczby i klasy.

*Por. np.: H. Oktaba, W. Ratajczak: *Simula 67*. Wydawnictwa Naukowo-Techniczne 1980.
**Norwegian Computing Center (Oslo)

Simula – schemat deklaracji klasy

Schemat deklaracji klasy w języku Simula:

```
class A(PA); WA; SA;
begin DA;
  I
end
```

gdzie:

- A – nazwa klasy;
- PA – lista parametrów formalnych;
- WA – zbiór wartości;
- SA – zbiór specyfikacji parametrów;
- DA – zbiór deklaracji lokalnych klasy;
- I – lista instrukcji.

Obiektowe języki programowania

Obiektowe języki programowania (ang. *object-oriented programming language*), to języki programowania wprowadzające pojęcia takie jak: obiekt, klasa, metoda, dziedziczenie, hermetyzacja i polimorfizm.

Przykładowo są nimi np.:

Smalltalk	C++
Eiffel	Java
Sather	CLOS
Ada95	OO-Cobol
Beta	Cecil
Dylan	Python
Self	Theta

i inne.

Zalety podejścia obiektowego

Podejście obiektowe jest powszechnie uznawane i stosowane, m.in. w wyniku następujących zalet:

- **spójność modeli systemu** – powiązanie fazy analizy i projektowania systemu;
- **łatwiejsza abstrakcja elementów dziedziny** – mechanizmy abstrakcji są niejako wbudowane w podejście obiektowe;
- **stabilność względem wprowadzanych zmian** – wprowadzenie zmian jest łatwiejsze i bezpieczniejsze;
- **możliwość wielokrotnego użycia komponentów** – poprzez dziedziczenie i uściślanie możliwość łatwego rozszerzania i redefiniowania komponentów;
- **skalowalność** – względnie luźne powiązanie elementów w połączeniu z mechanizmami abstrakcji i enkapsulacji ułatwia skalowalność systemu;
- **niezawodność i bezpieczeństwo** – mechanizmy podejścia obiektowego pozwalają na udostępnienie tylko tych interfejsów które są do tego przeznaczone;
- **wspieranie współbieżności** – w sposób naturalny wsparcie współbieżności poprzez definiowanie niezależnych komponentów.

Definicja obiektu

Obiekt jest jednym z najbardziej fundamentalnych pojęć we współczesnej informatyce (inżynierii oprogramowania). Pojęcie to i zagadnienia z nim związane są już dość dobrze rozpoznane, zarówno na gruncie teorii jak i praktyki.

DEFINICJA 3. *Obiektem* (ang. *object*) nazywamy abstrakcyjne pojęcie lub część otaczającego świata, wyróżniające się ograniczonym zakresem oraz interfejsem, za pośrednictwem którego obiekt komunikuje się z otoczeniem. ⌟

Każdy *obiekt* jest scharakteryzowany poprzez:

— *atrybuty* – odnoszą się do danych zamkniętych we wnętrzu obiektu, opisują jakby własności obiektu,

— ....

Definicja obiektu (cd.)

— *sposób zachowania* (ang. *behaviour*) – to akcje (działania) podejmowane przez obiekt. Zachowanie dzieli się na:

proste – dotyczy realizacji obsługi i uzależnione jest od wartości pewnej funkcji, a nie jest uzależnione od historii poprzednich żądań;

dyskretne – to zachowanie uzależnione jest od pojęcia stanu, a mianowicie obiekt reaguje na pewne zdarzenie w zależności od stanu w którym się znajduje, oraz zachowanie

ciągłe – oznacza to nieograniczony zbiór możliwych warunków, tak więc zachowanie takiego obiektu, w przeciwieństwie do zachowania dyskretnego, nie może być opisane przez automat, zachowanie obiektów jest uzależnione zarówno do historii jak i strumienia danych wejściowych (regulator).

Definicja obiektu (cd.cd.)

— *stan wewnętrzny* (ang. *state*) – wartości zmiennych składające się na strukturę obiektu, określone (przypisywane) w dyskretnych momentach czasowych.

— *zakres stosowania* (ang. *responsibility*) – uzależniony jest od kontekstu użycia obiektu – zachowanie powinno opisywać wszystkie możliwe zastosowania.

Uwagi odnośnie stanów:

— stan nie jest wprost tożsamy z wartościami atrybutów,

— stany możemy ustalać badając możliwe wartości atrybutów i sprawdzając czy zachowanie obiektu jest różne dla tych wartości.

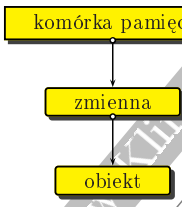
Zdarzenie to godna uwagi zmiana stanu.

Inna definicja obiektu

DEFINICJA 4. *Obiekt* (ang. *object*) to konkretny lub abstrakcyjny byt (wyróżnialny w modelowanej rzeczywistości) posiadający nazwę, jednoznaczną identyfikację, określone granice, atrybuty i inne własności. Obiekt może być – i zazwyczaj jest – skojarzony z metodami lub operacjami, które na nim działają; z reguły, są one definiowane/przechowywane w ramach jego klasy oraz jej nadklas. ⌟

Istnieje pewna różnica pomiędzy obiektami (realnymi) świata przedmiotu, a obiektami (abstrakcyjnymi) świata analizy i projektowania.

Rozwój pojęcia obiektu



komórka pamięci → adres + zawartość

zmienna → identyfikator + zawartość

obiekt → identyfikator + zawartość + operacje

Obiekt w zasadniczej swojej intencji jest dokładnie tym samym co zmienna dla języków programowania. Występujące różnice dotyczą rozwoju historycznego, np. w C++ występują zarówno zmienne (przejęte z C) jak i obiekty. Druga różnica polega na tym, że w większości klasycznych języków programowania nie istnieją mechanizmy pozwalające na związanie metod ze zmiennymi (czyli mechanizmy klas lub abstrakcyjnych typów danych).

Analiza, projektowanie i programowanie obiektowe OOA/OOD/OOP

Analiza – studiowanie dziedziny problemu, prowadzące do wyspecyfikowania zachowania zewnętrznie obserwowalnego. Określenie co jest potrzebne aby osiągnąć pożądaną funkcjonalność, łącznie z charakterystykami operacyjnymi (np. niezawodność, wydajność, itd.). *Analiza obiektowa* OOA (ang. *Object-Oriented Analysis*) wymaga stosowania do powyższej fazy paradygmatów podejścia obiektowego.

Projektowanie – przejęcie specyfikacji zewnętrznie obserwowalnej i określenie co należy zrobić, tj. jakie szczegóły są niezbędne, aby w sposób rzeczywisty osiągnąć wyspecyfikowaną funkcjonalność. *Projektowanie obiektowe* OOD (ang. *Object-Oriented Design*) związane jest ze stosowaniem do powyższej fazy paradygmatów podejścia obiektowego.

Programowanie – model stworzony w fazie projektowania podlega tłumaczeniu na odpowiedni język programowania z uwzględnieniem szeregu aspektów (np. środowisko wykonawcze, architektura sprzętowa i inne). *Programowanie obiektowe* OOP (ang. *Object-Oriented Programming*) związane jest ze stosowaniem do powyższej fazy paradygmatów podejścia obiektowego.

Obiekt, klasa, instancja

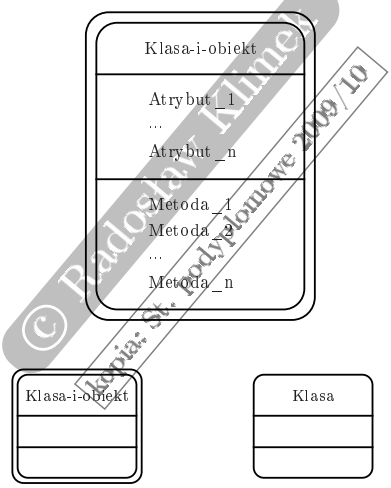
DEFINICJA 5. *Obiekt* (ang. *object*) to abstrakcja czegoś w dziedzinie problemu (identyfikacja, stan, zachowanie), ukazuje zdolności systemu do przechowywania o tym informacji oraz wykonywania na tym operacji. ⌟

DEFINICJA 6. *Klasa* (ang. *class*) to abstrakcja grupy obiektów o podobnej charakterystyce (wspólne własności, wspólne zachowania), jednolity zbiór atrybutów oraz jednolity zbiór usług łącznie ze sposobem tworzenia (inicjowania) obiektu. Klasa grupuje elementy względem pewnej relacji równoważnościowej.

Tak więc jeszcze inaczej można powiedzieć, że *klasa* jest zbiorem obiektów tego samego typu, które mogą się różnić wartością atrybutu lub zachowaniem, lecz pełnią tę samą rolę w systemie. ⌟

DEFINICJA 7. Odwoływanie się do każdego z elementów grupy obiektów jest odwoływaniem się do *instancji* (ang. *instance*) odpowiedniego obiektu, tj. do jego konkretnego wystąpienia (egzemplarza). Elementy klasy nazywamy instancjami. ⌟

Pierwsze notacje graficzne (wg. Coada-Yourdona)



Podstawowe operacje na obiektach

Wyróżnia się pięć podstawowych operacji – i ogólnie kategorii operatorów – działających na obiektach, składających się na jego charakterystykę zachowania:

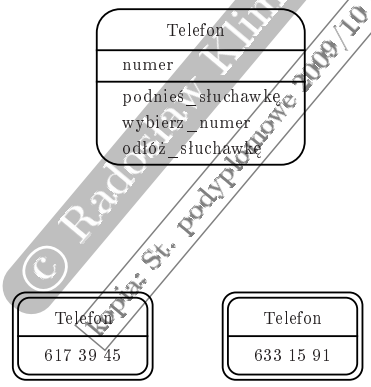
1. *konstruktor* (ang. *constructor*) – utworzenie obiektu wraz z zainicjowaniem jego zmiennych (zainicjowanie stanu początkowego);
2. *modyfikator* (ang. *modifier*) – zmiana stanu obiektu;
3. *selektor* (ang. *selector*) – udostępnienie informacji o stanie obiektu, bez dokonywania zmiany w obiekcie;
4. *iterator* (ang. *iterator*) – udostępnienie informacji o obiekcie poprzez dostęp do całej jego struktury, np. w wyniku iteracyjnego przeglądania struktury – przeglądanie informacji obiektu w dobrze (i ściśle) określonym porządku;
5. *destruktor* (ang. *destructor*) – niszczenie (usunięcie) obiektu.

Typy obiektów a operacje

Ze względu na rodzaje operacji podejmowanych przez obiekty i związki operacyjne pomiędzy nimi, wyróżnia się trzy rodzaje obiektów:

1. *aktorzy* (ang. *actors*) – obiekty które dokonują operacji na innych obiektach, lecz same nigdy nie podlegają operacjom ze strony innych obiektów;
2. *serwery* (ang. *serwers*) – obiekty podlegające operacjom ze strony innych obiektów i nie wykonujące operacji na innych obiektach;
3. *agenci* (ang. *agents*) – obiekty zarówno operujące na innych obiektach, jak i same podlegające działaniu ze strony obiektów innych.

Klasa i obiekt – przykład



Relacje podejścia obiektowego

Kategorie relacji w podejściu obiektowym

W klasycznym podejściu obiektowym wyróżnia się pięć podstawowych typów relacji występujących pomiędzy klasami/obiektami:

1. asocjacja (ang. association) lub skojarzenie,
2. agregacja (ang. aggregation),
3. kompozycja (ang. composition),
4. dziedziczenie (ang. inheritance) lub generalizacja,
5. uściślenie (ang. refinement).

W przypadku niektórych relacji można zuważyć bliskie podobieństwo, np. w przypadku relacji dziedziczenia i uściślania. Te dwie relacje opisują zależności pewnych linii generowania klas od specyfikacji bardziej ogólnych do bardziej szczegółowych. Jednak istnieje pewna istotna różnica pomiędzy tymi relacjami, a jest nią fakt, że dziedziczenie polega na dostosowaniu zachowania obiektów do określonych wymagań, natomiast uściślanie polega na precyzowaniu typu danych przy zachowanym tym samym (ogólnym) zachowaniu obiektu.

Relacja asocjacji

DEFINICJA 8. Relacja *asocjacji* (lub *skojarzenia*) (ang. *association*) oznacza możliwość przesyłania informacji pomiędzy obiektami. Fakt komunikacji ma miejsce wtedy, gdy jeden z obiektów korzysta z usług dostarczanych przez obiekt inny. ⌟

Uwagi i uzupełnienia:

— asocjacja może łączyć dwie lub więcej klas;

— obiekt korzystający z usług nie jest zawarty (agregacja) wewnątrz dostarczającego usługi;

— czas życia klasy używanej nie jest zakresem działania klasy użytkowników;

— relacja asocjacji wiąże słabiej niż relacja agregacji;

— przykładem relacji może być np. związek między pracownikiem i jego zakładem pracy;

— asocjacja umożliwia przechodzenie (nawigację) pomiędzy powiązanymi obiektami w dowolnym kierunku;

— relacja asocjacji może być określona jako relacja klient-serwer.

Relacja agregacji

DEFINICJA 9. Relacja *agregacji* (ang. *aggregation*) ma miejsce wtedy, gdy jeden obiekt zawiera inne. Obiekt zewnętrzny nazywany jest *właścicielem*, a obiekty zawarte (wewnętrzne) *komponentami*. ⌟

Uwagi i uzupełnienia:

— dopuszcza się także możliwość istnienia wspólnych komponentów przez kilku właścicieli;

— właściciel jest zazwyczaj uprawniony do kreowania i usuwania obiektów komponentów;

— agregacja może być rozpatrywana jako szczególny przypadek relacji asocjacji;

— agregacja modelnie stosunek całości do jej części (np. samolot i jego śmigła);

— obiekty są powiązane związkiem agregacji, jeżeli jeden z nich można uważać za część drugiego, zaś cykl i czas życia tych obiektów są jednakowe.

Relacja kompozycji

DEFINICJA 10. Relacja *kompozycji* (ang. *composition*) jest silniejszą odmianą relacji agregacji poprzez wprowadzenie następujących ograniczeń:

— komponenty nie mogą być dzielone pomiędzy wielu właścicieli,

— komponenty są powoływane i usuwane przez właściciela. ⌟

Relacja ta jest stosowana w sytuacjach, gdy pewne wątki są powoływane przez obiekt-właściciel, kóry grupuje korzeń wątków wykonywania, natomiast komponenty realizują poszczególne wątki.

Relacja dziedziczenia

DEFINICJA 11. Relacja *dziedziczenia* (ang. *inheritance*) lub *generalizacja* ma miejsce wtedy, gdy pomiędzy klasami obiektów zachodzi przekazywanie cech (definicji atrybutów, metod, itd.) z *nadklasy* do jej *podklas*.

Albo też równoważnie, dziedziczenie ma miejsce wtedy, gdy dana klasa może być zdefiniowana z wykorzystaniem charakterystyk innej klasy, będącej *Klasą-rodzicem*, z ewentualnym dodaniem dodatkowych atrybutów lub rozszerzeniem zachowań. ⌟

Uwagi i uzupełnienia:

- bardzo efektywny mechanizm definiowania nowych klas z wykorzystaniem rodziców, tzn. potomek dziedziczy cechy rodzica, za wyjątkiem tych cech które są określone na nowo;
- mechanizm wyrażania podobieństwa pomiędzy klasami, pozwalający na upraszczanie definicji klas podobnych do już zdefiniowanych;
- przykład: obiekt klasy „Student" dziedziczy wszystkie własności (definicje atrybutów, metody) określone w ramach klasy „Osoba";
- istnieje wiele form dziedziczenia, np. dziedziczenie oparte na prototypach lub delegacja, dziedziczenie dynamiczne, wielokrotne dziedziczenie, itd.

Relacja dziedziczenia (cd.)

Dziedziczenie to podstawowy mechanizmem sprzyjającym ponownemu użyciu.

Dziedziczenie jest realizowane w poszczególnych językach i systemach na wiele różnych sposobów, a mianowicie możemy mieć do czynienia z dziedziczeniem pojedynczym lub prostym (od jednego rodzica), np. Ada, Smalltalk, albo też dziedziczeniem wielokorotnym, np. C++. Dziedziczenie przebiega zazwyczaj wzdłuż pojedynczej charakterystyki, względnie niewielkiego zbioru charakterystyk.

Dziedziczenie dynamiczne (ang. dynamic inheritance) – dziedziczenie przez obiekt cech klas lub cech innych obiektów (np. ich stanu) podczas czasu wykonania; często powiązane z możliwością dynamicznej zmiany dziedziczonych cech lub dynamicznej zmiany przynależności obiektu do klasy. Dziedziczenie dynamiczne można osiągnąć wyłącznie w językach/systemach, gdzie klasa jest obywatelem pierwszej kategorii lub w językach/systemach opartych o koncepcję prototypu.

Dziedziczenie statyczne (ang. static inheritance) – dziedziczenie, które można rozstrzygnąć podczas kompilacji.

Relacja uściślania

DEFINICJA 12. Relacja *uściślania* (ang. *refinement*) to relacja umożliwiająca niekompletne specyfikowanie elementów (tutaj klas), które mogą być konkretyzowane poprzez dodanie elementów wcześniej nieokreślonych. Niekompletna specyfikacja stanowi definicję wyższego poziomu, będąc rodziną klas nieprzydatną do bezpośredniego użycia, a dopiero po uściśleniu definicja staje się kompletna i może być stosowana w odpowiedniej konstrukcji. ⌟

W poszczególnych językach takie niekompletne specyfikacje nazywają się odmiennie, np. w C++ używane jest pojęcie szablonu (ang. template), natomiast w Adzie jest to jednostka rodzajowa (ang. generic), bardzo dokładnie uściślanie zostało zdefiniowane w języku UML.

Identyfikowanie klas i obiektów

Identyfikowanie klas obiektów na przykładzie pewnego opisu problemu:

> „System rezerwacji sprzedaży biletów na przedstawienia w różnych teatrach"

Wstępnie można tu wyróżnić następujące klasy obiektów:

- System,
- Rezerwacja,
- Bilet,
- Przedstawienie,
- Teatr.

W kolejnym kroku szuka się związków pomiędzy nimi poprzez odnajdywanie podobieństw, co prowadzi do eliminacji niewłaściwych klas obiektów wg. odpowiednich reguł.

Reguły eliminowania klas

Przykładowe reguły eliminacji niewłaściwych klas obiektów:

1. redukowanie klas namiarowych – gdy powstają klasy o różnych nazwach, lecz dotyczące tego samego rodzaju obiektu;
2. redukowanie klas nieistotnych – usunięcie klas nie mających związku z opisywanym problemem (decyzja raczej arbitralna);
3. poprawianie klas źle określonych – każda klasa powinna mieć dobrze określone granice swojego występowania;
4. redukowanie klas będących cechami – nazwy klas mogą opisywać własności;
5. redukowanie klas będących operacjami – podobnie jak powyżej należy dokonać redukcji, gdyż operacje określane będą jako sekwencje w tzw. modelu dynamicznym;
6. stosowanie odpowiednich nazw klas – nazwa klasy powinna odzwierciedlać naturę obiektu, a nie jego rolę jaką obiekt odgrywa w systemie;
7. eliminowanie klas obiektów implementacyjnych – klasy muszą odzwierciedlać obiekty rzeczywiste, a nie implementacyjne, takie np. jak: procesory, procedury itd.

Metodologia Coada-Yourdona

Metodologia Coada-Yourdona

Metodyka Coada-Yourdona, zwana czasem metodyką OOA/OOD

— wyróżnia pięć aktywności: wyodrębnienie klas i obiektów, wyodrębnienie struktur, wyodrębnienie podmiotów lub dziedzin, określenie atrybutów, określenie usług.

— Składowymi metody są: interakcja z ludźmi, określenie dziedziny problemu, zarządzanie zadaniami, zarządzanie danymi.

— OOA/OOD jest uważana za metodykę dość złożoną i opartą na wyrafinowanych pojęciach.

Wyróżnia się dwie (a raczej trzy) tzw. struktury, odpowiadające podstawowym relacjom podejścia obiektowego:

1. struktura generalizacji-specjalizacji – podobieństwo pomiędzy klasami poprzez uszczegółowianie i uogólnianie;
2. struktura całość-część – wyróżnianie obiektów zarówno jako całości, jak i ich składowych;
3. kojarzenie – wskazywanie powiązań pomiędzy obiektami na podstawie cech obiektów (atrybuty).

Analiza obiektowa w ujęciu Coada-Yourdona

Analiza obiektowa OOA (ang. *Object-Oriented Analysis*) wg. Coada-Yourdona składa się z pięciu kroków, czasami zwanych także warstwami analizy:

1. warstwa klas-obiektów;

2. warstwa atrybutów;

3. warstwa metod (usług);

4. warstwa struktur;

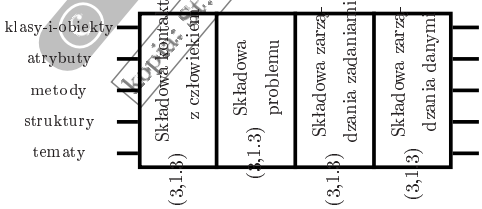
5. warstwa tematów.

Składowe projektowania obiektowego

Projektowanie obiektowe OOD (ang. *Object-Oriented Design*) posiada cztery składowe:

1. składowa dziedziny problemu;
2. składowa kontaktu z człowiekiem;
3. składowa zarządzania zadaniami;
4. składowa zarządzania danymi.

Na powyższe składowe nakładają się znane warstwy analizy obiektowej.

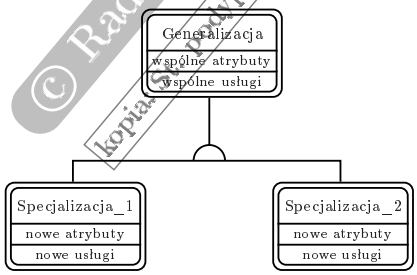


Struktura generalizacja-specjalizacja

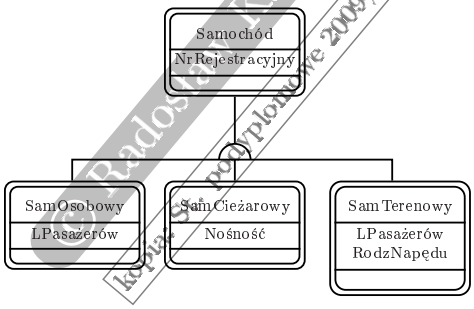
Struktura *generalizacja-specjalizacja* (lub skrótowo *gen-spec*), jest połączeniem dwóch przeciwstawnych podejść:

1. generalizacji – utworzenie klasy ogólniejszej w ten sposób, że dany obiekt należy także do wszystkich klas bardziej ogólnych, zwanych *nadklasami*;
2. specjalizacji – utworzenie *podklasy* z danej klasy.

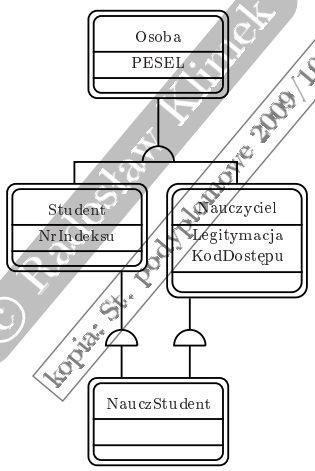
Struktury gen-spec modelują dziedziczenie – jednorazowe zdefiniowanie atrybutów lub metod, a następnie ich powielenie. Dziedziczenie dotyczy definicji atrybutów lub metod, a nie ich wartości.



Generalizacja-specjalizacja – przykład



Generalizacja-specjalizacja jako krata



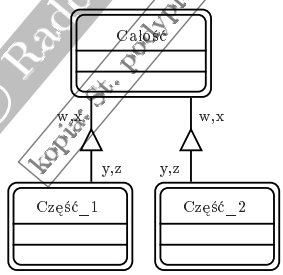
Struktura całość-część

Struktura *całość-część* jest typowa dla ludzkiego podejścia do rozwiązywania i organizacji problemów.

— Struktura całość-część modeluje relację agregacji.

— Struktura całość-część dotyczy obiektów i umożliwia wychwycenie ograniczeń dla danej dziedziny zastosowań.

— Ze strukturami całość-część związane są liczebności składowych stuktury.



Całość-część – przykłady