POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Modelowanie Systemów Informacyjnych (MSI)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 1

Obiektowość

http://www.mtrzaska.com

Informacje

- Tematyka wykładu
  - Modelowanie z użyciem notacji UML
  - Projektowanie
  - Elementy implementacji

- Zaliczenie
  - Kolokwia
  - Mini-projekty
  - Projekt

Informacje (2)

- Literatura

Mariusz Trzaska:

Modelowanie i implementacja systemów informatycznych. Rok 2008. Wydawnictwo PJWSTK. Stron 299. ISBN 978-83-89244-71-3.

Fragmenty: http://www.mtrzaska.com/mas-ksiazka

  - Książki o modelowaniu, UML, obiektowości.
  - Książki dotyczące programowania.

Informacje (3)

- Literatura – c. d, np.:
  - Ewa Stemposz, Andrzej Jodłowski, Alina Stasiecka: *Zarys metodyki wspierającej naukę projektowania systemów informacyjnych*. Wydawnictwo PJWSTK, 2013. ISBN 978-83-63103-39-2

  - ✓ J. Płodzień, E. Stemposz: Analiza i projektowanie systemów informatycznych, wydawnictwo PJWSTK, 2003 i wydanie II-gie rozszerzone 2005;
  - ✓ Śmiałek: *Zrozumieć UML 2.0 Metody modelowania obiektowego*, Helion;
  - ✓ S. Wrycza, B. Marcinkowski, K. Wyrzykowski: *Język UML 2.0 w modelowaniu systemów informatycznych*, Helion.

Zagadnienia

Metodyki, notacje i UML

Geneza obiektowości

Obszary oddziaływania obiektowości:

- obiektowe metodyki
- obiektowe języki programowania
- obiektowe bazy danych

Przeszkody dla obiektowości

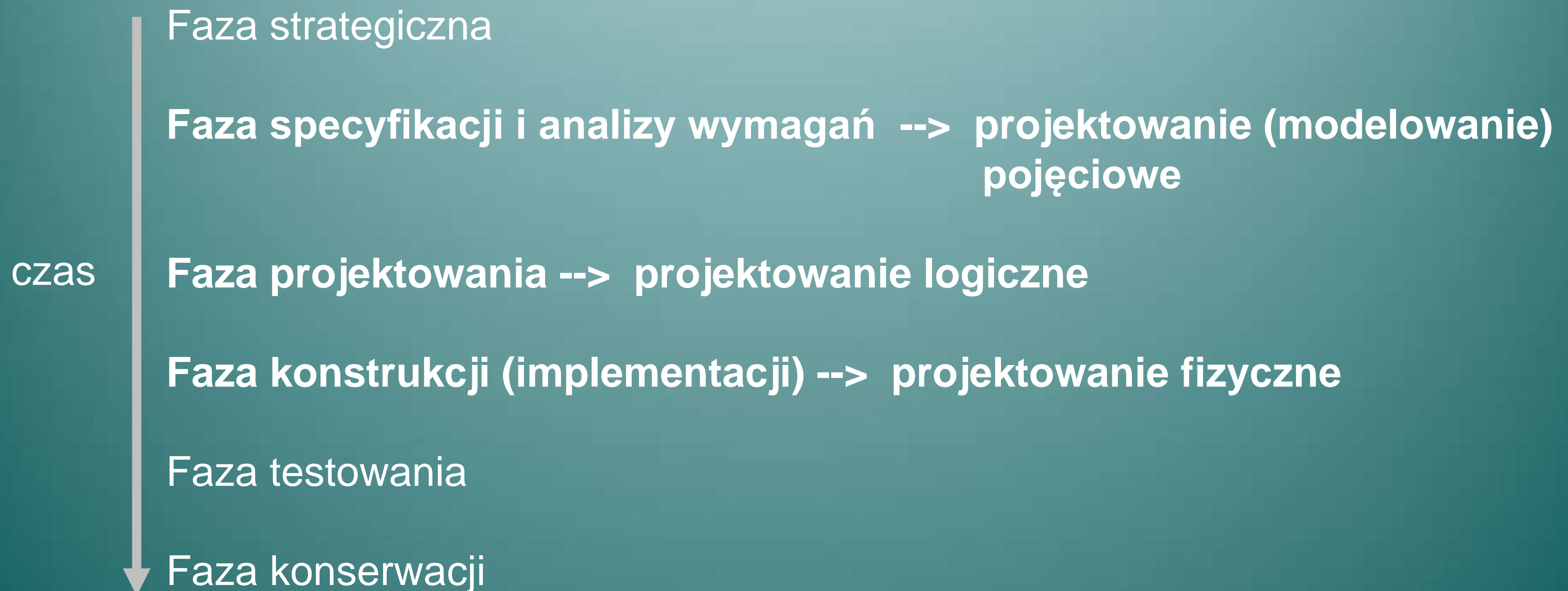
Podstawowe zasady obiektowości:

- obiekt
- tożsamość obiektu
- hermetyzacja
- klasa
- dziedziczenie
- polimorfizm

Wykorzystano materiały z wykładu PRI autorstwa dr inż. Ewy Stemposz oraz prof. Kazimierza Subiety

Model wytwarzania produktu informatycznego

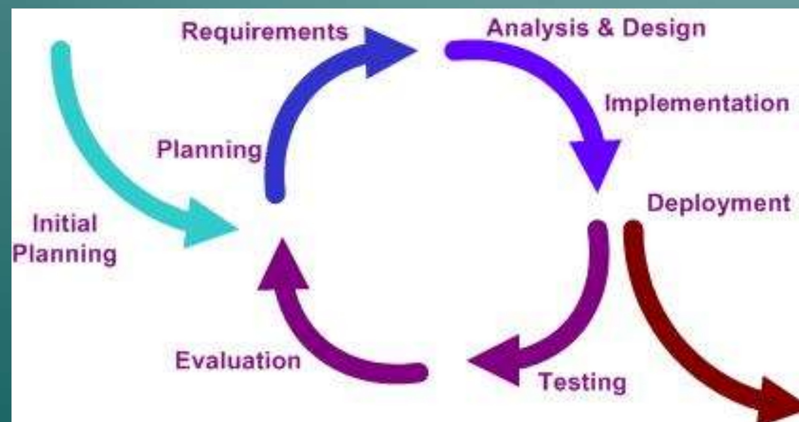
Fazy cyklu życiowego produktu informatycznego w klasycznym modelu kaskadowym:



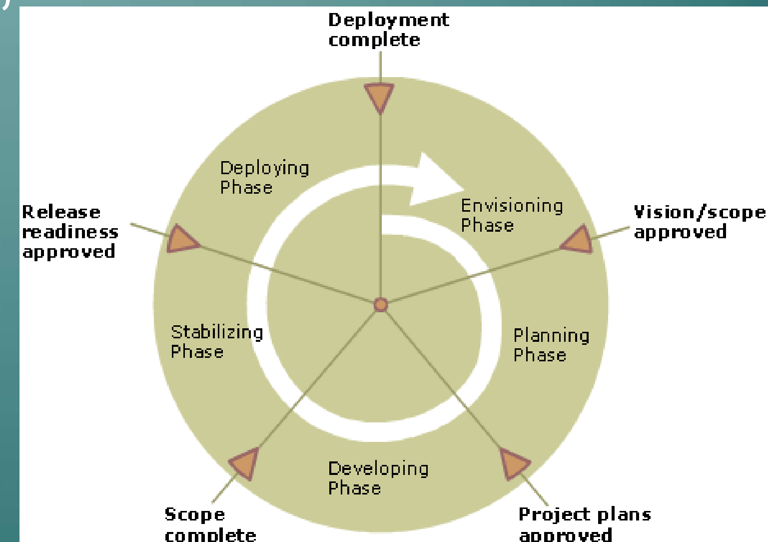
Obecnie stracił na popularności na rzecz różnych metodyk iteracyjnych

Model wytwarzania produktu informatycznego (2)

- Obecnie najpopularniejsze są różne metodyki iteracyjne oraz *agile* (lekkie, zwinne):
  - Metodyka spiralna,
  - MSF (Microsoft Solution Framework),
  - Extreme Programming (XP),
  - Scrum



Źródło: Wikipedia



Źródło: Microsoft

Modelowanie pojęciowe

- Pojęcia: modelowanie pojęciowe (conceptual modeling) oraz model pojęciowy (conceptual model) odnoszą się do procesów myślowych, do wyobrażeń towarzyszących pracy nad oprogramowaniem.

- Projektant, programista, itd. muszą dokładnie wyobrazić sobie problem oraz metodę jego rozwiązania. Zasadnicze procesy tworzenia oprogramowania zachodzą więc w ludzkim umyśle i nie są związane z jakimkolwiek językiem programowania czy jakimkolwiek narzędziem w ogóle.

- Modelowanie pojęciowe może być (i powinno być) wspomagane przez środki wzmacniające ludzką pamięć i wyobraźnię, służące do opisu odwzorowywanej rzeczywistości w postaci: struktur danych, operacji na danych czy zachodzących procesów.

Metodyka (1)

Metodyka (metodologia) – w inżynierii oprogramowania – jest zestawem pojęć, oznaczeń, języków, modeli, diagramów, technik i sposobów postępowania wspierających proces konstruowania systemu (realizacji projektu).

Metodyka jest wykorzystywana zarówno do projektowania pojęciowego, jak i logicznego czy fizycznego.

Metodyka ustala fazy realizacji projektu, a ponadto dla każdej z faz projektu wyznacza:

- role uczestników projektu,
- scenariusze postępowania,
- reguły przechodzenia do następnej fazy,
- modele, które powinny być wytworzone,
- dokumentację, która powinna powstać,
- język/notację, którą należy używać.

Metodyka (2)

Notacja, czyli zbiór oznaczeń, jest wykorzystywana do dokumentowania wyników poszczególnych faz projektu – pośrednich i końcowych. Notacja służy jako środek wspomagający ludzką pamięć i wyobraźnię, a także jako środek ułatwiający komunikację zarówno między członkami zespołu projektowego, jak i między zespołem projektowym a klientem.

Rodzaje notacji:

- tekstowa
- specyfikacje - ustrukturalizowany zapis tekstowy i numeryczny
- notacje graficzne

Szczególne znaczenie mają notacje graficzne, ich zalety potwierdzają badania psychologiczne. Inżynieria oprogramowania wzoruje się tu na innych dziedzinach techniki, takich jak np. elektronika czy mechanika.

Dana notacja może być wykorzystywana w wielu metodykach.

Język do modelowania

Język do modelowania, jak każdy inny język, oprócz *notacji, semantyki* i *składni* posiada jeszcze jeden ważny aspekt: *pragmatykę*.

Semantyka określa, co należy rozumieć pod przyjętymi oznaczeniami (notacją).

Składnia określa, jak wolno łączyć ze sobą przyjęte oznaczenia.

Pragmatyka określa, w jaki sposób należy używać przyjętych oznaczeń, jak do konkretnej sytuacji dopasować pewien wzorzec notacyjny – zgodnie z intencją autorów języka.

Język niewiele znaczy bez znajomości sposobów wykorzystywania go w procesie wytwarzania produktu programistycznego. W metodykach pragmatyka stosowanego języka jest sprawą podstawową. Jest ona zazwyczaj trudna do objaśnienia: można to robić wyłącznie na przykładach przypominających realne sytuacje. Niestety, realne sytuacje są zazwyczaj bardzo skomplikowane, czego efektem jest pewien infantylizm przykładów zamieszczanych w podręcznikach.

Unified Modeling Language (UML)

RATIONAL SOFTWARE CORPORATION http://www.rational.com/uml (*dokumentacja on line*)

UML 0.8 styczeń 1995 - wrzesień 1996
UML 1.0, styczeń 1997, przesłany do OMG
UML 1.1, koniec 1997, zatwierdzony jako składnik standardu OMG
UML 1.3, kwiecień 1999
UML 1.4 listopad 2001
…
UML 2.5.1 2017-12
https://www.omg.org/spec/UML/

Połączone siły trzech znanych metodologów oprogramowania:

Grady Booch

Ivar Jacobson

James Rumbaugh

UML: Krótka charakterystyka (1)

- UML cieszy się aktualnie bardzo dużą popularnością. Prawdopodobnie przez wiele najbliższych lat będzie dominował w obszarze analizy i projektowania.
- UML jest metodyką projektowania ?

Definicja podana przez Rational: "The Unified Modeling Language (UML) is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system."

- Notacja UML, która opiera się o podstawowe pojęcia obiektowości może być wykorzystana w dowolnej metodyce.
- Pojęcia UML, wynikające z doświadczenia jej twórców, mają w założeniu przykrywać większość istotnych aspektów modelowanych systemów.
- UML jest składową standardu OMG (CORBA).
- Nie wszyscy są zachwyceni UML. Niektórzy specjaliści uważają go za twór przereklamowany: niestabilny, zbyt ciężki, źle zdefiniowany. UML ma konkurentów w postaci metodyki i notacji OPEN, “design by contracts” oraz innych.

Diagramy definiowane w UML

Diagramy przypadków użycia (*use case*)

Diagramy klas
Diagramy dynamiczne (*behavior*):

- Diagramy stanów
- Diagramy aktywności
- Diagramy interakcji:
  Diagramy sekwencji
  Diagramy współpracy (*collaboration*)

Diagramy implementacyjne:

- Diagramy komponentów
- Diagramy wdrożeniowe (*deployment*)

Diagramy pakietów

Diagramy te zapewniają uzyskanie wielu perspektyw projektowanego
systemu w trakcie jego budowy.

Model a diagram; modele w UML

Model: pewna abstrakcja projektowanego systemu, widziana z określonej perspektywy, na określonym poziomie szczegółowości.

Diagram: środek służący do opisu modelu. Dany model może być opisany przy pomocy wielu diagramów. Dany element modelu może pojawiać się na wielu diagramach jednego modelu.

Dwa najważniejsze modele w UML, wykorzystywane w fazie analizy, to:

- Model przypadków użycia opisujący system widziany z perspektywy jego przyszłych użytkowników (za pomocą diagramów przypadków użycia).
- Model obiektowy przedstawiający statyczną budowę (strukturę systemu) za pomocą diagramów klas i diagramów obiektów. Diagram klas może zawierać obiekty. Diagram obiektów nie zawiera klas, ale wyłącznie obiekty.

Głównym zadaniem pomocniczego modelu dynamicznego (zachowań) jest wypełnienie diagramu klas metodami wynikłymi z analizy zachowania systemu w trakcie wykonywania zadań, gdzie zadaniem może być np. realizacja przypadku użycia czy też jednego konkretnego scenariusza danego przypadku użycia.

Model a diagram; modele w UML (2)

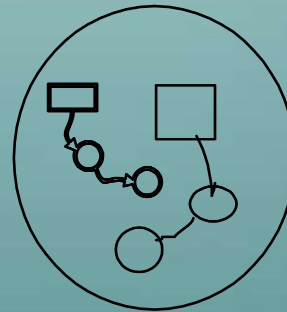
Główny obszar działania	Model	Diagramy	Podstawowe pojęcia
struktura	model obiektowy	diagram klas	klasa, obiekt, asocjacja, generalizacja, zależność, realizacja, interface
		diagram obiektów	
	model przypadków użycia	diagramy przypadków użycia	aktor, przypadek użycia, «include», «extend», generalizacja
	model implementacji	diagramy komponentów	komponent, interface, zależność, realizacja
		diagramy wdrożeniowe	węzeł, komponent, zależność, lokacja
dynamika	model dynamiczny	diagramy stanu	stan, zdarzenie, przejście, akcja, aktywność

Model a diagram; modele w UML (3)

Główny obszar działania	Model	Diagramy	Podstawowe pojęcia
dynamika, c.d.	model dynamiczny, c.d.	diagramy aktywności	stan, aktywność, fork, join, romb decyzyjny
		diagramy interakcji	interakcja, współpraca, komunikat, aktywacja
zarządzanie	model zarządzania	diagramy pakietów	pakiet, podsystem
rozszerzalność	wszystkie modele	wszystkie diagramy	stereotyp, własność etykietowana, ograniczenie

Geneza obiektowości

Obiektowość jest ideologią (początki sięgają lat 60-tych), która wynika z zaobserwowanych wad istniejącego świata i podaje jakąś receptę, jak te wady usunąć, a więc przede wszystkim stara się o uzyskanie jak najmniejszej luki pomiędzy myśleniem o rzeczywistości (dziedzinie problemowej), a myśleniem o danych i procesach, które zachodzą na danych.



Mentalna percepcja świata rzeczywistego

Model pojęciowy

Schemat struktury danych

W modelu relacyjnym model pojęciowy stara się odwzorować świat rzeczywisty, lecz jest ograniczony dostępną bazą implementacyjną. W rezultacie, schemat struktury danych gubi semantykę danych. Model obiektowy podtrzymuje te zgodności, przybliżając semantykę danych do świata rzeczywistego.

Źródła obiektowości

Języki programowania operujące na złożonych strukturach danych, wprowadzające klasy, metody, dziedziczenie i hermetyzację (Simula 67, Smalltalk).

Skierowanie uwagi na czynniki ludzkie w tworzeniu oprogramowania.

Źródła obiektowości

Metodyki projektowania oprogramowania, od początku bazujące na wyróżnianiu obiektów i ich klas w otaczającej nas rzeczywistości.

Bazy danych, od początku bazujące na obiektach (IMS, CODASYL).

Obszary oddziaływania obiektowości

- **Metodyki analizy i projektowania SI** (Rumbaugh, Booch, Jacobson, Yourdon,...) i oparte o nie **narzędzia CASE**. Najbardziej istotna zmiana w stosunku do metodyk wykorzystujących model encja-związek to możliwość związania z obiektami operacji, które można na nich wykonywać.

- **Języki programowania** (Smalltalk, C++, Java, Eiffel,...)
Klasy, dziedziczenie, hermetyzacja, metody, późne wiązanie.

- **Bazy danych i składy trwałych obiektów**
(standard ODMG- 2.0, ObjectStore, O2, Poet, Versant, ...)
Przeniesienie obiektowych technologii programowania na grunt baz danych.

- **Współdziałanie** systemów heterogenicznych (OMG CORBA, OLE/DCOM/ActiveX)
Obiekty i klasy jako podstawa wymiany informacji pomiędzy systemami.

- **Wizyjne** środowiska programistyczne (Smalltalk, CA OpenRoad, IBM VisualAge,...) Przeniesienie technik obiektowych do programowania wizyjnego.
- **Inne:** biblioteki oprogramowania, grafika, miary i oceny oprogramowania, re-inżynieria biznesu (BPR)

Obiektowe języki programowania (1)

Smalltalk

Język zbudowany w latach 1976-83 w Xerox Palo Alto Research Center w Kalifornii. Zawiera meta-klasy, klasy, podklasy, wirtualne funkcje, przesyłanie komunikatów. *Wszystko* jest tu obiektem, a w szczególności liczby i klasy. Istotą sukcesu Smalltalk'a jest to, że nie jest on tylko językiem prgramowania, lecz ponadto i mocnym, zintegrowanym środowiskiem programistycznym z doskonałym interfejsem okienkowym. Prostota, elastyczna natura i możliwość szybkich dynamicznych zmian uczyniła Smalltalk'a doskonałym narzędziem do łatwego tworzenia prototypów. Mniej są znane przemysłowe aplikacje na dużą skalę.

C++

Język hybrydowy, pochodna języka C. Łączy własności niskiego poziomu (przejęte z C), jak np. arytmetyka wskaźników, z konstrukcjami wysokiego poziomu, takimi jak klasy, podklasy, hermetyzacja, funkcje wirtualne. (Eklektyczna natura C++ jest przedmiotem krytyki.) Duże zastosowania na skalę przemysłową. Jednocześnie, jest on krytykowany z powodu wolnego tworzenia aplikacji, słabej przenaszalności, dużego ryzyka wadliwego działania programów.

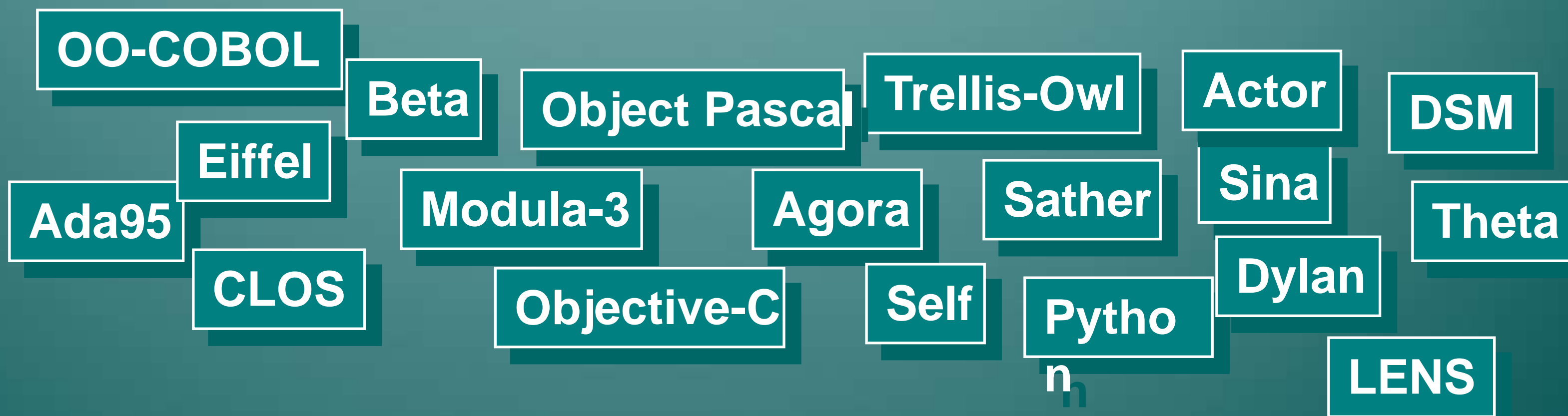
Obiektowe języki programowania (2)

Java

C#

Mieszanina C++, Smalltalk'a i Objective-C, z obcięciem własności niskiego poziomu. Język pomyślany jako narzędzie do programowania stron Webu (jedno z zastosowań). Istotną własnością języka Java jest to, że programy są kompilowane nie do poziomu kodu maszynowego, a do poziomu znakowego języka pośredniego, interpretowanego następnie za pośrednictwem maszyny wirtualnej. Daje to efekt dużej przenaszalności programów oraz zwiększenia bezpieczeństwa (*security*), co jest szczególnie istotne w środowiskach rozproszonych, takich jak np. Internet.

Ponadto mrowie języków:



Trwałość

Trwała wartość to taka, która żyje dłużej niż czas działania wykorzystującego ją programu − innymi słowy, wartość trwała przenosi się pomiędzy kolejnymi uruchomieniami tego programu.

Wszystko, co zawierają bazy danych, jest trwałe.

Trwała zmienna: zmienna programistyczna, która ma wszystkie własności normalnej zmiennej (w sensie konstrukcji programistycznych, w których może być użyta), ale której wartość przy nowym uruchomieniu programu jest taka sama, jak przy zakończeniu poprzedniego uruchomienia programu.

Popularne języki programowania (C, C++, Smalltalk, Pascal, Java,...) nie mają trwałych zmiennych. Wymagają one wczytania *explicite* wartości trwałej zmiennej z pliku zewnętrznego na swoją zmienną (i zapisania *vice versa*).

Istnieje grupa prototypowych języków posiadających trwałe zmienne (PJama).

Trwały obiekt: obiekt o własnościach trwałej zmiennej, obiekt bazy danych.

Ortogonalna trwałość

Tradycyjnie, **bazy danych** przechowywały **typy trwałe i masowe** (zbiory, relacje, etc.). Tradycyjnie, **języki programowania** zajmowały się **typami indywidualnymi i nietrwałymi** (zmienne, struktury, zapisy, etc.). Tradycyjnie, istnieją różnice w koncepcjach dostępu do bazy danych i dostępu do zmiennych programu.

Nie istnieje logiczne uzasadnienie takiego podziału. Można podać wiele przykładów, kiedy przydałoby się zapamiętanie w bazie danych jakichś zmiennych indywidualnych (np. nazwisko prezydenta RP). Podobnie, brak typów masowych w językach programowania doprowadził do koncepcji "sterty" *(heap),* która ma liczne wady, a w szczególności: ograniczoną kontrolę typów, konieczność dynamicznych operacji alokacji i zwalniania pamięci, konieczność przetwarzania poprzez wskaźniki.

Ortogonalna trwałość oznacza własność języka programowania polegającą na tym, że cecha trwałości jest ortogonalna do konstruktorów typu. W szczególności, baza danych może przechowywać dane indywidualne i trwałe, zaś w obszarze danych programu mogą znajdować się wartości masowe i nietrwałe.

Przeszkody dla obiektowości

Każda nowa ideologia ściera się z zastanym stanem rzeczy i poprzednimi ideologiami.

Z czym walczy obiektowość?

- Zastany świat interfejsów programistycznych (C, COBOL, Fortran, SQL, ...)
- Mity i fałszywe steoretypy:
  - Relacyjna baza danych zapewnia prostotę struktur danych.
  - Bezpośrednie powiązania (wskaźniki) w bazie danych są niekorzystne.
  - Tylko relacyjna baza danych zapewnia możliwość definiowania języków zapy
  - Tylko relacyjna baza danych zapewnia sprawne przetwarzanie transakcji.
  - Relacyjne bazy danych mają solidne podstawy matematyczne.
  - Relacyjne bazy danych mają bardzo dobrą wydajność, nieosiągalną dla innyc
- "Spuścizna": ogromne inwestycje w hierarchiczne, sieciowe i relacyjne bazy danych.
- Własne słabości: słabo wyartykułowane zasady, zbyt dużo formalizmów, różne języki, brak standardów.

Obiektowość - potencjalne ryzyko

- Niedopracowane mechanizmy zarządzania dużą bazą obiektów, sterowania wersjami, rejestrowania zmian.
- Technologie obiektowe są jak dotąd stosowane przez małe i średnie organizacje. Nie jest do końca pewne jak przeskalują się dla wielkich organizacji. Duża liczba tematów znajduje się ciągle w fazie laboratoryjnej. Szereg technologii jest mało stabilnych (np. metodyki projektowania).
- Przejście na technologie obiektowe może zagrozić funkcjonowaniu obecnie działających i sprawnych systemów, które są krytyczne dla misji organizacji.
- Zbyt mała liczba ekspertów jest wyszkolona w zakresie technologii obiektowych.
- Nie jest jasne, jakie koszty pociągnie za sobą przejście na technologie obiektowe.
- Standardy w zakresie obiektowości są niedopracowane i niestabilne. Nie wiadomo w jakim zakresie będą one pełnić swoją funkcję.

Obiekt

Rzecz lub pojęcie obserwowalne w tym fragmencie świata rzeczywistego, którego dotyczy dany system informacyjny (dziedzinie problemowej), posiadające nazwę oraz dobrze określone granice, jest odwzorowywane na **obiekt** w implementacji komputerowej. Pojęcie obiektu sprzyja lepszemu rozumieniu modelowanego świata rzeczywistego - byty ze świata rzeczywistego odpowiadają obiektom w programie.

Obiektem może być np. pewien **zamknięty fragment oprogramowania** (dana, procedura, moduł, dokument, okienko dialogu,...), którym można operować jak zwartą bryłą: usuwać, wyszukiwać, wiązać, kopiować, blokować, indeksować, ...

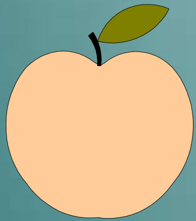
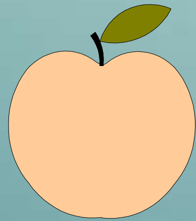
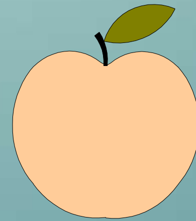
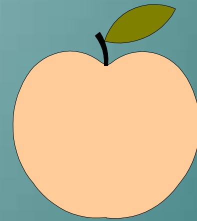
1. Obiekt ma przypisany **typ**, tj. wyrażenie językowe, które określa jego budowę (poprzez specyfikację atrybutów) oraz ogranicza kontekst, w którym odwołanie do obiektu może być użyte w programie.

2. Obiekt może być **złożony**, tj. może składać się z innych obiektów.

3. Obiekt może być **powiązany** z innymi obiektami związkami skojarzeniowymi (powiązaniami), odpowiadającymi relacjom zachodzącym między odpowiednimi bytami w dziedzinie problemowej.

Identyfikator obiektu

Byt jest wyróżnialny w otaczającym nas świecie poprzez fakt swojego istnienia, a nie przez jakąkolwiek własność, która odróżnia go od innych bytów.

Może się zdarzyć, że z punktu widzenia naszych obserwacji (tj. postrzegania charakterystycznych własności) dwa byty są nieodróżnialne, np. rodzeństwo bliźniąt. Niemniej jednak są to dwie **różne** osoby.

W implementacji komputerowej system odwzorowywuje tożsamość bytu w identyfikator − unikalny dla każdego obiektu − nadawany automatycznie. Identyfikator jest atrybutem **„wewnętrznym”** obiektu, nie ma żadnego znaczenia dla dziedziny problemowej. Programista/użytkownik nigdy nie operuje explicite jego wartością.

(1) Mechanizm identyfikatorów pozwala zarówno na rozróżnianie obiektów, jak i umożliwia budowanie do nich powiązań.

(2) Identyfikator może być trwały, tj. niezmienny w trakcie całego życia obiektu.

Własności obiektu

Obiekt jest charakteryzowany poprzez:

1

Tożsamość, która odróżnia go od innych obiektów. Tożsamość obiektu jest niezależna zarówno od wartości atrybutów czy powiązań obiektu, jak i od lokacji bytu odwzorowywanego przez obiekt w świecie rzeczywistym czy też od lokacji samego obiektu w przestrzeni adresowej komputera.

(W praktyce: tożsamość = trwały **wewnętrzny identyfikator** obiektu)

2

Stan, który może zmieniać się w czasie (bez zmiany tożsamości obiektu). Stan obiektu w danym momencie jest określony przez aktualne wartości jego atrybutów i powiązań z innymi obiektami.

3

Obiekt ma przypisane **zachowanie**, tj. zestaw operacji, które wolno stosować do danego obiektu.

Przykład obiektu

Wpłać

Wypłać

Sprawdź stan

Porównaj podpis

Numer = 1234567
Stan konta = 34567 PLN
Właściciel = Jan Kowalski
Upoważniony = ...
Podpis = …
....

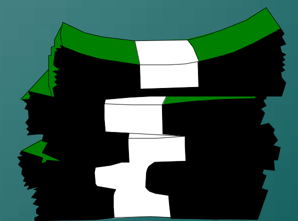
Nalicz procent

Zlikwiduj konto

Upoważnij

Podaj osoby upoważnione

Obiekt
KONTO



Relatywizm obiektów

- Obiekt reprezentujący pewien byt świata rzeczywistego powinien zawierać wewnątrz siebie wszelkie informacje, które odnoszą się do tego bytu.

- Informacje opisujące byt, są odwzorowywane w obiekcie poprzez mechanizm atrybutów − każdy obiekt opisywany jest przez pewną liczbę atrybutów, które także mogą być złożone.

- Atrybuty obiektu mogą być obiektami („podobiekty”).

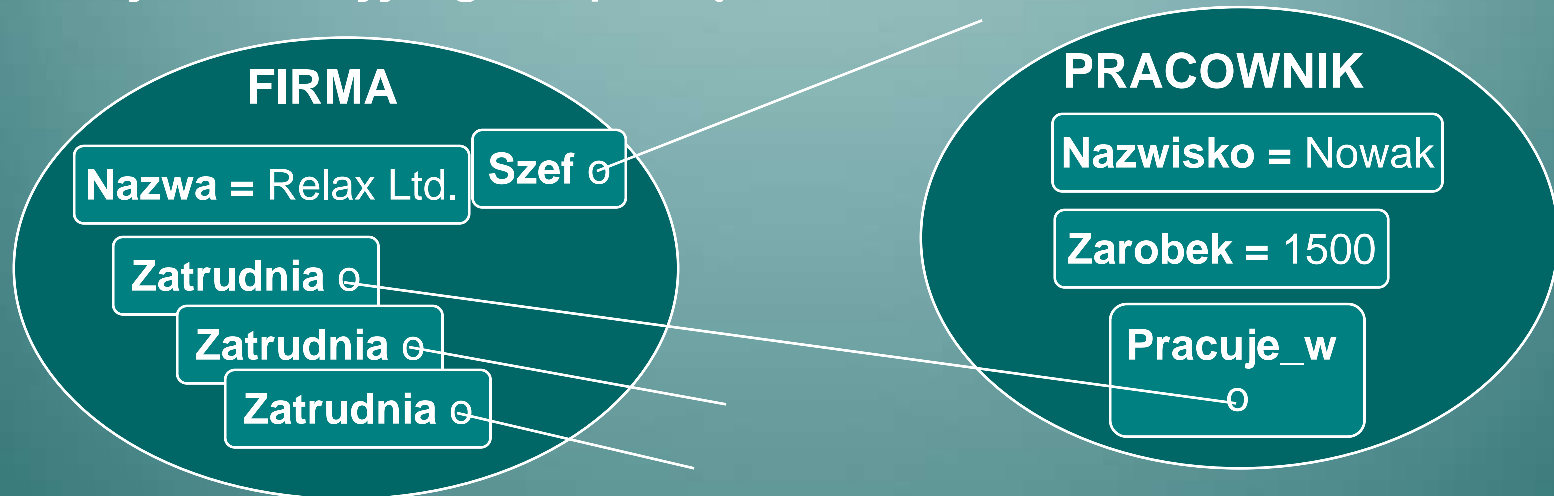
- Nie powinny występować ograniczenia na liczbę/rozmiary atrybutów opisujących obiekt, liczbę poziomów hierarchii podobiektów, co w efekcie powinno prowadzić do nie istnienia ograniczeń na rozmiary obiektów.

- Ustalenia, które informacje odnoszą się do danego obiektu, a które do innego, zależą od modelu pojęciowego analityka i nie powinny podlegać ograniczeniom ze strony środowiska realizacyjnego.

Przykład obiektu złożonego

Powiązania pomiędzy obiektami

Możliwe jest tworzenie bezpośrednich powiązań prowadzących od jednego obiektu do innego. Powiązanie jest daną zbudowaną w oparciu o identyfikator obiektu. Unika się tu pojęcia wskaźnika, na rzecz czegoś "bardziej abstrakcyjnego" − powiązania.



Zalety powiązań: naturalne odwzorowanie semantycznych związków istniejących w dziedzinie problemowej między analizowanymi bytami poprzez powiązania między obiektami; łatwe nawigowanie dzięki wyrażeniom ścieżkowym; zwiększenie szybkości działania.

Wady: zwiększona "sztywność" struktury danych; możliwość utraty spójności wskutek "zwisających" powiązań; możliwość naruszenia reguł hermetyzacji.

Klasa; dwie niezbyt zgodne definicje

1. Klasa jest **nazwanym zbiorem obiektów** o podobnych własnościach (podobna semantyka, podobne atrybuty, zachowania, podobne związki z innymi obiektami). Własności te są określone w definicji klasy. Stosunek klasa/podklasa oznacza zawieranie się zakresów znaczeniowych. Np. zbiór obiektów **Student** zawiera się w zbiorze **Osoba**.

2. UML: Klasa jest **nazwanym opisem grupy obiektów**, które współdzielą ten sam zbiór własności (inwariantów). Klasa nie jest zbiorem obiektów, lecz jest używana do opisywania (deklarowania) obiektów. Stosunek klasa/podklasa oznacza, że obiekty podklasy posiadają wszystkie inwarianty nadklasy, plus (ewentualnie) inwarianty swoje. Np. klasa **Student** ma wszystkie inwarianty klasy **Osoba**, plus inwarianty własne.

Najważniejsze inwarianty to:

Nazwa, czyli językowy identyfikator klasy obiektu
Typ, opis struktury (budowy) obiektu − poprzez atrybuty
Metody, zbiór operacji, które można wykonać na obiekcie

Możliwe są inne inwarianty:

Zdarzenia lub wyjątki,
Obsługa zdarzeń lub wyjątków (reguły aktywne)
Lista eksportowa określająca, co jest dostępne z zewnątrz
Ograniczenia, którym może podlegać obiekt klasy
. . .

Metody jako przykład inwariantów klasy

Zwykle, mamy do czynienia z wieloma obiektami tej samej klasy, np. z wieloma kontam

Nie jest celowe, aby każdy z obiektów danej klasy przechowywał w sobie własną kopię metod lub informacji o swoim typie (budowie). Ta informacja jest przechowywana w jednym miejscu, w klasie.

Obiekty KONTO

Klasa wszystkich kont

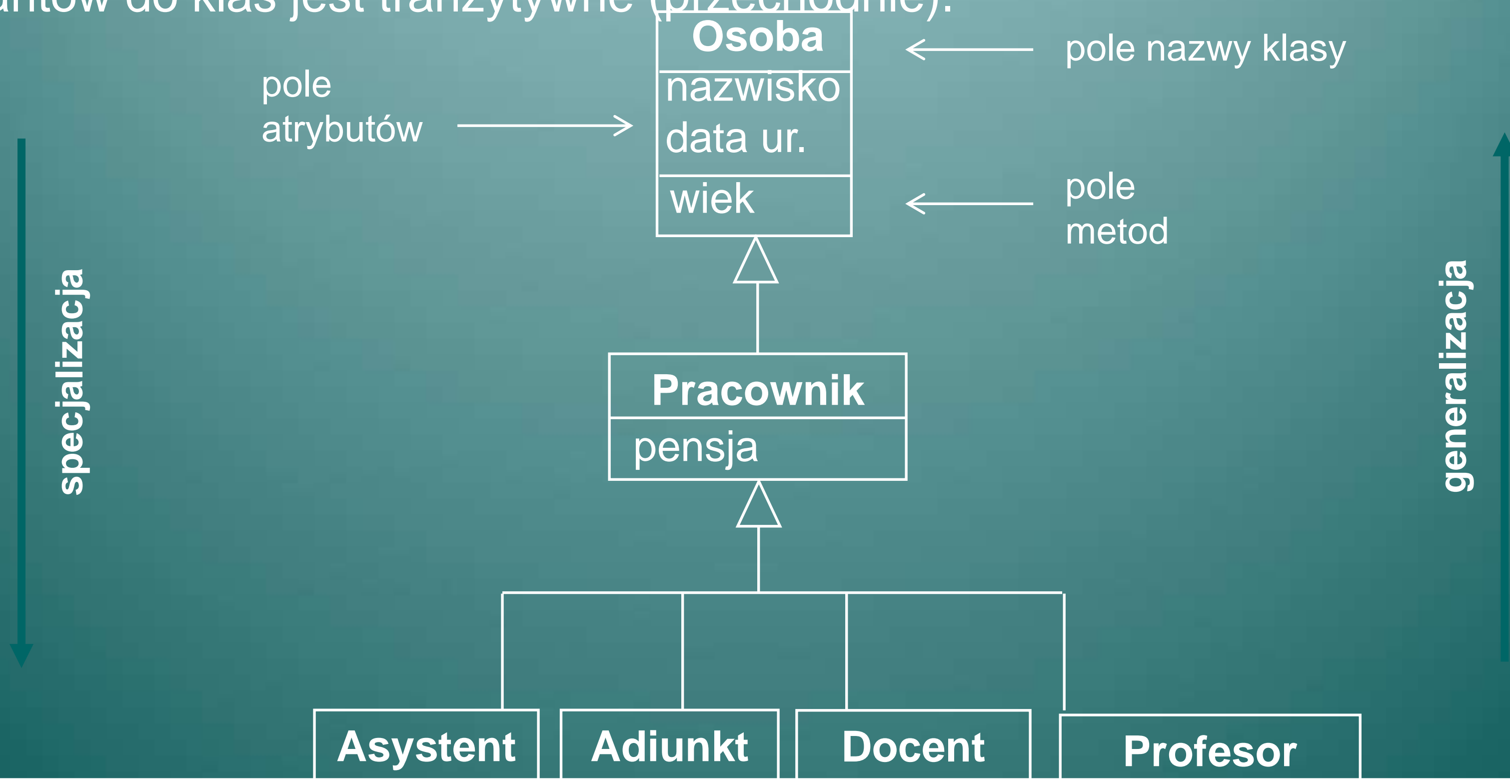
import inwariantów

Numer = 1234321
Stan konta = 34567
Właściciel = Jan Kowalski
Upoważniony =
. . .

Numer = 1234567
Stan konta = 454545
Właściciel = Adam Nowak
Upoważniony =
. . .

Generalizacja-specjalizacja a dziedziczenie

Generalizacja-specjalizacja jest takim związkiem pomiędzy klasami, który łączy klasę bardziej ogólną (nadklasę) z jedną lub więcej klas (tzw. podklas), będących jej specjalizacjami. Klasa, będąca specjalizacją danej klasy, oprócz własności nadklasy może posiadać (i z reguły posiada) też własności swoje. Związek generalizacji-specjalizacji może być implementowany za pomocą struktur dziedziczenia, co nie jest jedynym możliwym rozwiązaniem. Dziedziczenie inwariantów do klas jest tranzytywne (przechodnie).



Hermetyzacja; ukrywanie informacji

Hermetyzacja: zgromadzenie elementów struktury i implementacji obiektu w postaci jednej manipulowalnej bryły; **oddzielenie specyfikacji obiektu od jego implementacji.** Hermetyzacja pośrednio oznacza także **ukrycie** struktury i implementacji obiektu. Tę własność określa się jako **ukrywanie informacji**. Hermetyzacja i ukrywanie informacji są różnymi pojęciami, choć mocno powiązanymi.

Zasada inżynierii oprogramowania (Parnas, 1972): programista ma tyle wiedzieć o obiekcie programistycznym, ile potrzeba, aby go efektywnie użyć. **Wszystko, co może być przed nim ukryte, powinno być ukryte.**

Hermetyzacja i ukrywanie informacji są podstawą pojęć: modułu, klasy i ADT.

Hermetyzacja **ortodoksyjna** (Smalltalk)

Na zewnątrz są widoczne metody; atrybuty obiektu są ukryte.

Ergo: prawie każdy atrybut *atr* jest obsługiwany przez dwie metody: *czytaj_atr, zmień_atr*

Hermetyzacja **ortogonalna** (C++)

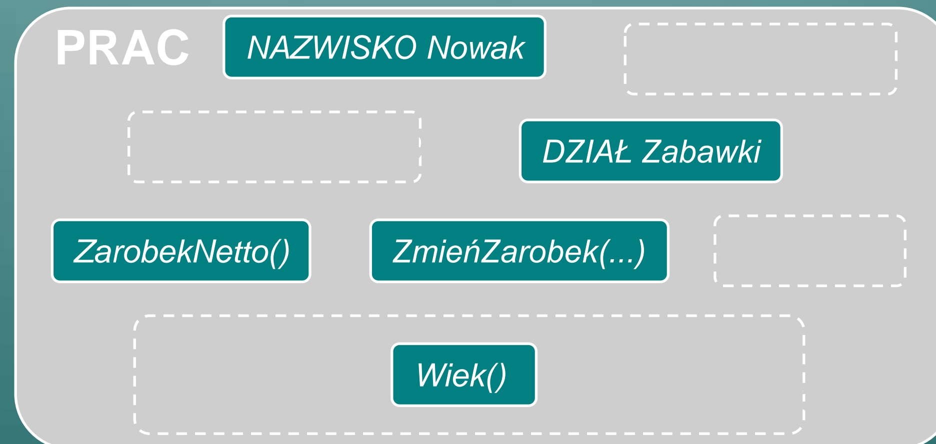
Dowolna własność obiektu (atrybut, metoda,...) może być prywatna (ukryta) lub publiczna

Specjalne środki do specyfikowania własności prywatnych i publicznych.

Hermetyzacja ortogonalna

Patrz Modula-2: dowolna własność może być prywatna, lub może być "wyeksportowana" do publicznego użytku.

Wewnętrzna struktura obiektu



Zewnętrzna struktura obiektu

Operacja a metoda (1)

Operacja jest funkcją, która może być zastosowana do obiektu. Operacja jest własnością klasy obiektów.

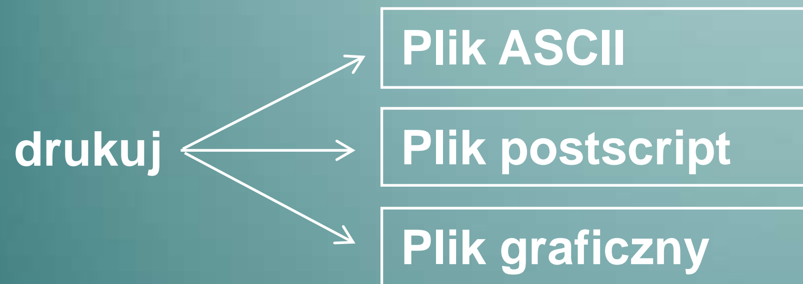
zatrudnij
zwolnij
wypłać dewidendę
} możliwe **operacje** na obiektach klasy **Pracownik**

Wszystkie obiekty, będące członkami danej klasy, podlegają tym samym operacjom. <u>Dana operacja może być stosowana do obiektów wielu różnych klas, połączonych związkiem generalizacji-specjalizacji.</u>

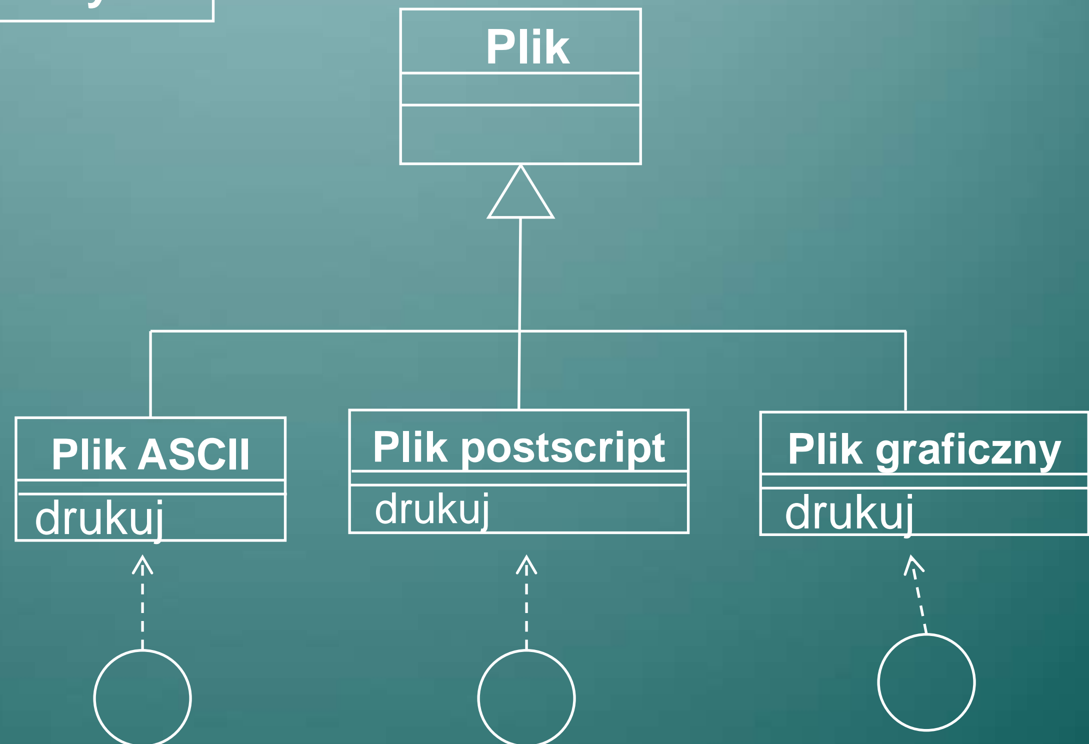
Metoda jest implementacją operacji w jednej z klas połączonych związkiem generalizacji-specjalizacji, co oznacza, że może być wiele metod implementujących daną operację.

<u>**operacja: co?**</u>
<u>**metoda: jak?**</u>

Operacja a metoda (2)



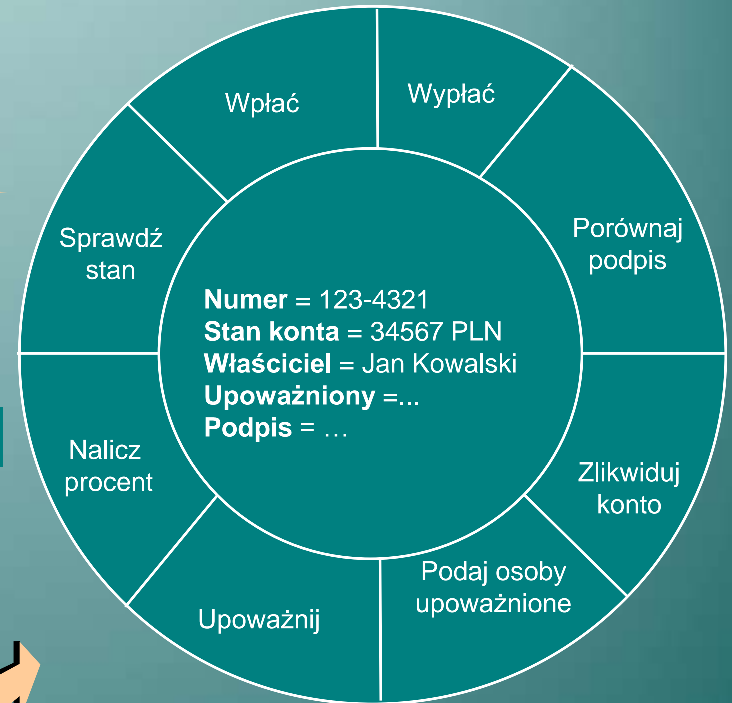
Jedna operacja **drukuj**, ale różne sposoby drukowania − **trzy metody** implementujące operację drukuj**.**



Komunikat (2)

Wypłać 1000 PLN

OK, wypłaciłem



Graj

Cccco proszę...?

Komunikat a wołanie funkcji (1)

Komunikat: obiekt-adresat poprzedza wywołanie operacji:
obiekt.operacja (arg1, arg2, ...) − obiekt jest tu domyślnym argumentem metody

Wołanie funkcji: obiekt jest komunikowany jako jeden z argumentów funkcji:
funkcja (obiekt, arg1, arg2, ...)

Nie jest to wyłącznie różnica syntaktyczna, gdyż:

Dla metody, środowisko na którym działa, może zmieniać się dynamicznie (późne wiązanie metod, w odróżnieniu od wczesnego wiązania dla funkcji).

Komunikat nie określa, **która** z metod implementujących daną operację ma być wywołana; wysłanie komunikatu do konkretnego obiektu powoduje wywołanie metody, implementującej żądaną operację, związanej z danym obiektem.

Przykład: operacja **drukuj** wykonywana iteracyjnie dla zbioru plików o różnych formatach.

Komunikat a wołanie funkcji (2)

```
X = dochody(emeryt)
Y = dochody(pracownik)
```

Przełączanie, związane z rodzajem obiektu, następuje w ciele funkcji **dochody**. Funkcję zwykle pisze jeden programista, na wszystkie okazje. Każda zmiana, związana z nowym rodzajem obiektów, wymaga zmian w ciele funkcji.

```
X = emeryt.dochody()
Y = pracownik.dochody()
```

Nie ma przełączania; za każdym razem, w zależności od rodzaju obiektu − adresata komunikatu, wywoływana jest inna metoda **dochody**. Obie metody implementują operację **dochody**. Obie metody (i ich programiści) nie muszą nic o sobie wiedzieć.

Polimorfizm (1)

Z greckiego, **polimorfizm** − oznacza „wiele form” („wiele postaci”) jednego bytu. Słowo “polimorfizm” też jest polimorficzne, istnieje co najmniej kilka rodzajów polimorfizmu, zgodnie z poniższą specyfikacją:

- **Polimorfizm metod** − (co zostało już wyjaśnione wcześniej w punkcie „Operacja a metoda”) polega na tym, że operacja wywoływana za pośrednictwem komunikatu może być różnie wykonana, w zależności od rodzaju obiektu, do którego ten komunikat został wysłany; innymi słowy może istnieć wiele metod implementujących daną operację.
- **Polimorfizm typów** (z teorii typów) − polimorfizm w tzw. polimorficznych językach programowania − oznacza istnienie funkcji, które mogą zarówno przyjmować wartości wielu typów jako swoje argumenty, jak też i zwracać wartości wielu typów.

Przykładowo, funkcja *daj_pierwszy(lista)* zwraca pierwszy element dowolnej listy, niezależnie od tego, czy jest to lista liczb całkowitych, czy lista liczb rzeczywistych, czy lista rekordów, czy też inna. Polimorfizm typów jest uważany za podstawę programowania ogólnego (ang. generic). Temat pozostaje jednak w strefie akademickiej, gdyż języki z polimorfizmem typów (a w szczególności ML) uważane są za zbyt wyrafinowane dla przeciętnego programisty.

Polimorfizm (2)

Dość powszechne jest **plątanie polimorfizmu metod z polimorfizmem typów**. Argumentacja, że polimorfizm metod jest szczególnym przypadkiem polimorfizmu typów (obiekt, do którego jest wysłany komunikat jest dodatkowym argumentem metody) jest niezbyt przekonywująca.

Polimorficzne języki programowania (ML, Quest, Napier88,...**) nie muszą być obiektowe, ale mogą być obiektowe (**Fibonacci**).**

- **Polimorfizm parametryczny.** Rodzaj polimorfizmu typów, który oznacza, że typ bytu programistycznego może być parametryzowany innym typem, np. klasa WEKTOR (int) czy WEKTOR (char).

Obiektowość a redukcja złożoności (1)

Potencjał obiektowości dla potrzeb redukcji złożoności oprogramowania:

- **Modelowanie świata rzeczywistego** jest ułatwione dzięki zastosowaniu podejścia obiektowego. Klasy, grupujące obiekty odwzorowywujące byty ze świata rzeczywistego, są najbardziej stabilnym elementem dziedziny problemowej. Doświadczenie wykazuje, że oprogramowanie oparte o klasy powstałe w wyniku analizy dziedzinowej jest bardziej odporne na zmiany wymagań niż oprogramowanie skonstruowane w oparciu o jednostki funkcjonalne.

- **Hermetyzacja** wspomaga redukcję złożoności poprzez zachęcanie konsumentów, by opierali się raczej na interfejsie do obiektu niż na jego wewnętrznej organizacji. Abstrahowanie od szczegółów implementacyjnych znacząco ułatwia proces rozumienia. Ponadto, hermetyzacja pozwala na ukrycie poprawek czy modyfikacji przed konsumentem oprogramowania.

Obiektowość a redukcja złożoności (2)

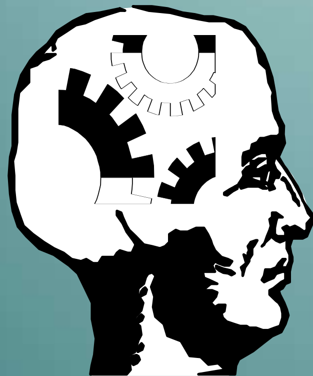
- **Dziedziczenie** pozwala na specjalizowanie struktur i zachowania obiektów podklasy bez ingerowania w struktury i zachowania obiektów nadklas. Poprzez dostarczenie opisu wyjaśniającego zasady organizacji struktury dziedziczenia, można pośrednio wpływać na jej racjonalny rozwój, nawet nie zawsze w kierunku przewidzianym przez jej twórcę.

- **Polimorfizm metod** wspiera redukcję złożoności pozwalając, by nowe bardziej wyspecjalizowane komponenty mogły być wykorzystywane w tym samym środowisku, co mniej wyspecjalizowane, bez potrzeby zmiany środowiska przy każdej zmianie komponentów.

- **Polimorfizm parametryczny** wspomaga redukcję złożoności umożliwiając definiowanie rodziny klas o takim samym interfejsie i implementacji, różniących się jedynie typem wyspecyfikowanym jako parametr klasy, np. klasa WEKTOR(int) czy WEKTOR(char).

Podsumowanie

Obiektowość jest ideologią, która zmienia myślenie realizatorów SI z "zorientowanego na maszynę" na "zorientowane na człowieka".



- Obiektowość jest konsekwencją kryzysu oprogramowania: kosztów związanych z oprogramowaniem, jego zawodnością i trudną do opanowania złożonością.
- Obiektowość przenika wszelkie fazy projektowania, narzędzia i interfejsy.
- Obiektowość dopracowała się własnej kolekcji pojęć i narzędzi.
- Obiektowość jest na początku swojej drogi i musi walczyć z konserwą i spuścizną poprzednich ideologii.