

Projektowanie systemów obiektowych

Projektowanie systemu i projektowanie klas

Krzysztof Goczyła

Wojciech Waloszek

kris@eti.pg.gda.pl

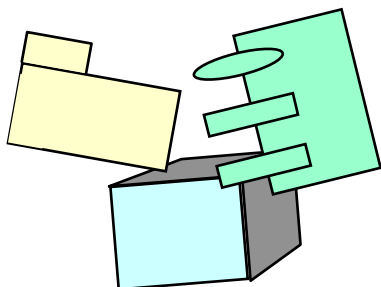
wowal@eti.pg.gda.pl

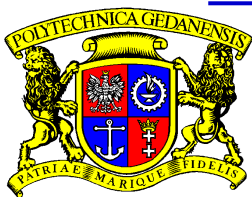
Katedra Inżynierii Oprogramowania

Wydział Elektroniki, Telekomunikacji i

Informatyki

Politechnika Gdańska





Przedmiot – warunki zaliczenia

Wykład + laboratorium = 100%

Wykład = 50%

Laboratorium + Projekt = 50%

Zaliczenie przedmiotu wymaga zaliczenia wykładu, laboratorium i projektu!

Zasady zaliczenia wykładu: zalicza 51% z egzaminu, który odbędzie się na końcu semestru.

Zasady zaliczenia laboratorium i projektu: podane na zajęciach.

Ocena:

<51%, 60%) dostateczna,

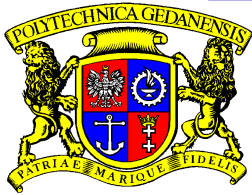
<60%, 70%) dostateczna plus,

<70%, 80%) dobra,

<80%, 90%) dobra plus,

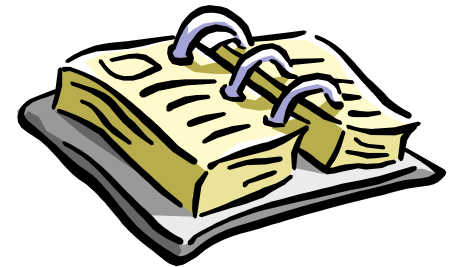
<90%, 100%) bardzo dobra,

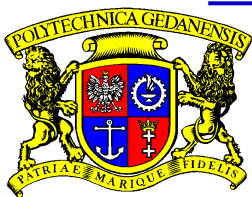
<100%, ∞) celująca.



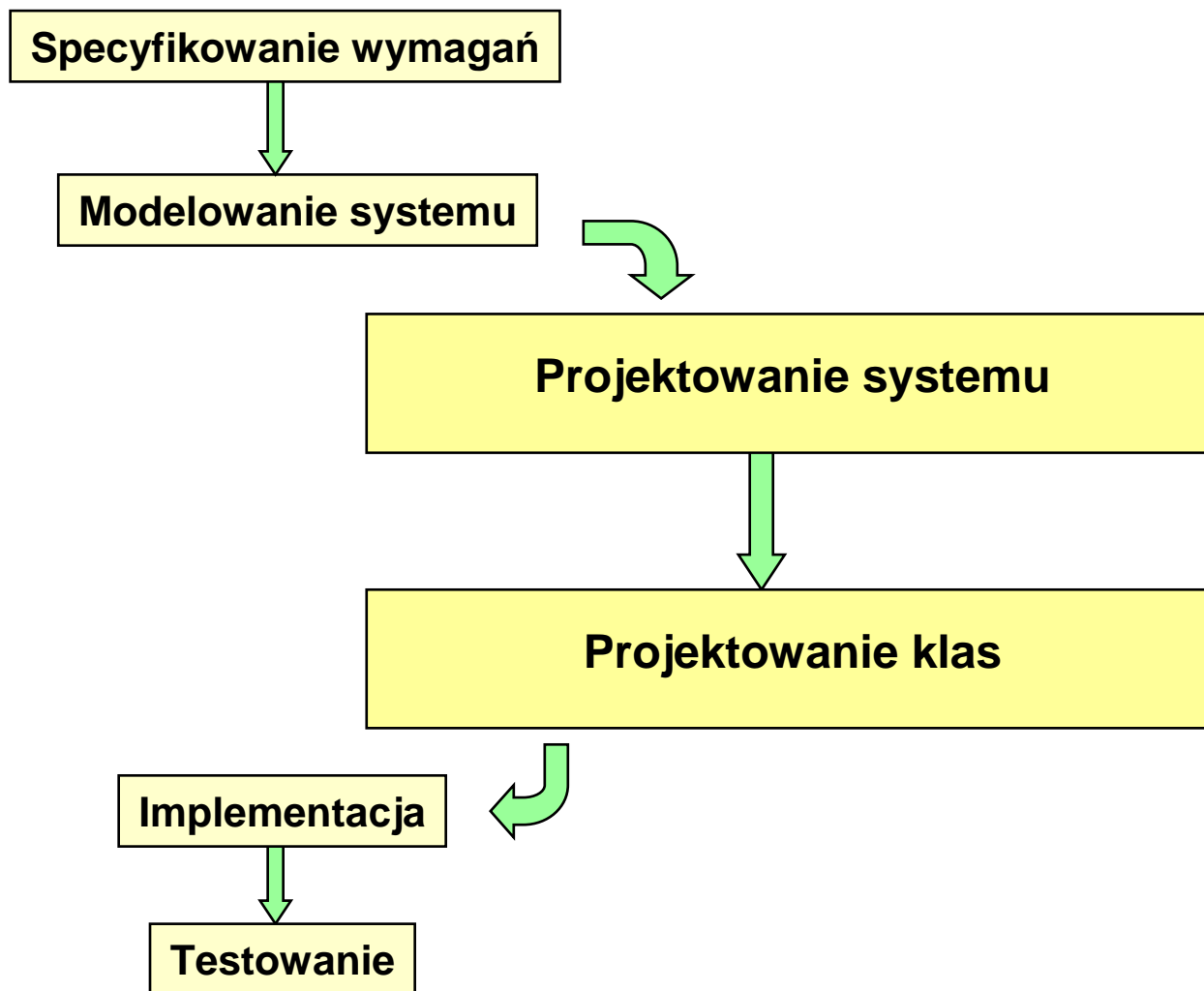
Wybrana literatura

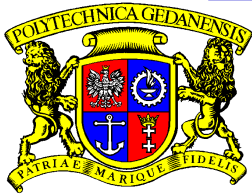
- Booch G., Rumbaugh J., Jacobsen I.: *UML - Przewodnik użytkownika*. WNT, 2001
- Dumnicki R. Kasprzyk A., Kozłowski M. *Analiza i projektowanie obiektowe*, Helion, 1998
- Górski J. (red.) *Inżynieria oprogramowania w projekcie informatycznym, wyd. 2 rozszerzone*, Mikom, 2000.
- Jaskiewicz A. *Inżynieria oprogramowania*, Helion, 1997.
- Subieta K. *Obiektość w projektowaniu i bazach danych*, Akademicka Oficyna Wydawnicza PLJ, 1998.
- Szejko S. (red). *Metody wytwarzania oprogramowania*. Mikom 2002.
- Szyperski C.: *Oprogramowanie komponentowe – obiekty to za mało*. WNT 2001
- UML Objectory: www.rational.com, www.omg.org
- Alur D, Crupi J. Malks D. *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, Sun, 2001
- Cooper J. W. *Java Wzorce projektowe*. Helion.2001
- Gamma E, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Software Architecture*, Addison-Wesley, 1995.
- Fowler M, *Analysis Patterns: Reusable Object Models*, 1997
- Roman, E. Ambler S., Jewell T. *Mastering Enterprise JavaBeans*, John Wiley & Sons, 2001





Etapy obiektowego wytwarzania oprogramowania

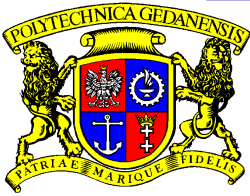




Projektowanie systemu

- **Czynności**
 - Wewnętrzna organizacja systemu: określenie jego podsystemów
 - Identyfikacja współbieżności zawartej w systemie
 - Przydzielenie podsystemów do procesorów
 - Wybranie podejścia dla zarządzania pojemnikami danych
 - Identyfikacja i obsługa zasobów globalnych
 - Obsługa stanów przejściowych i awaryjnych
 - Wybór priorytetów
 - Ogólny projekt interfejsów systemu

- **Wyniki**
 - Podstawowa architektura systemu
 - Decyzje strategiczne dotyczące systemu

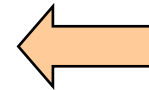


Podział na podsystemy

Podsystem jest zbiorem klas, związków, zdarzeń i ograniczeń, które są ze sobą silnie powiązane, a ich interakcja z innymi elementami systemu jest stosunkowo niewielka.

• Podział ze względu na:

- wspólny zestaw oferowanych usług
- wspólne umiejscowienie
- podobny sprzęt (platforma)



Źródła przesłanek podziału:

- dziedzina aplikacyjna
- ograniczenia zasobów
- środowisko pracy
- logiczna architektura systemu

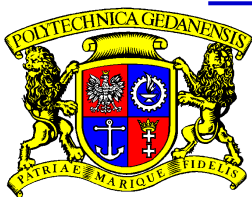
Przykład: system komputerowy statku kosmicznego może składać się z następujących podsystemów:

- podtrzymania życia
- nawigacyjny
- sterowania silnikami
- eksperymentów naukowych
- kontaktu z operatorem (interfejs)



Liczba podsystemów?

- Zależy od złożoności problemu
- Kompromis między kosztem interfejsów a łatwością pielęgnacji

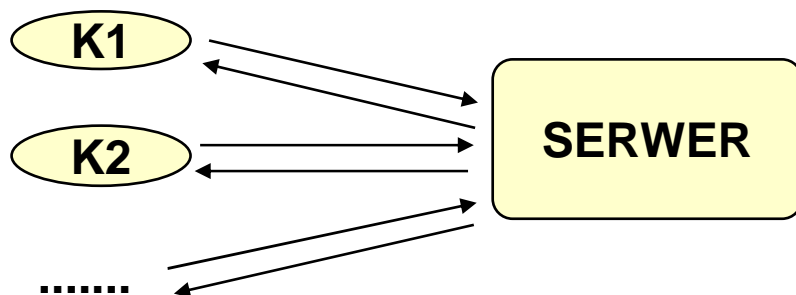


Podział na podsystemy

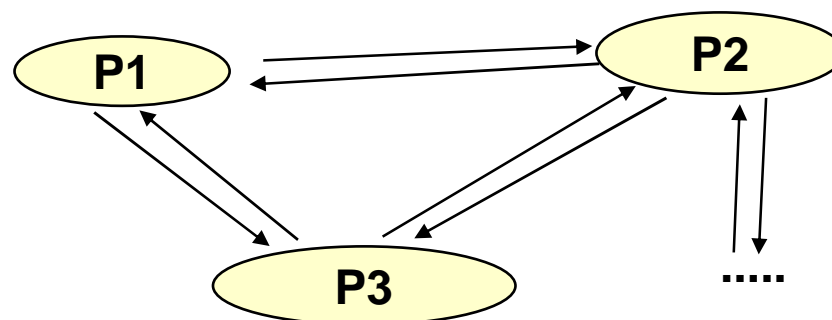
- Podsystemy zwykle są identyfikowane przez **zakres** dostarczanych **usług**
- Każdy podsystem ma **interfejs** służący do komunikacji z resztą systemu

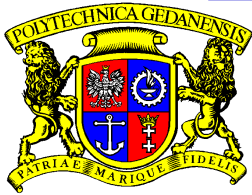
Możliwe związki z innymi podsystemami:

klient - serwer (client-server)



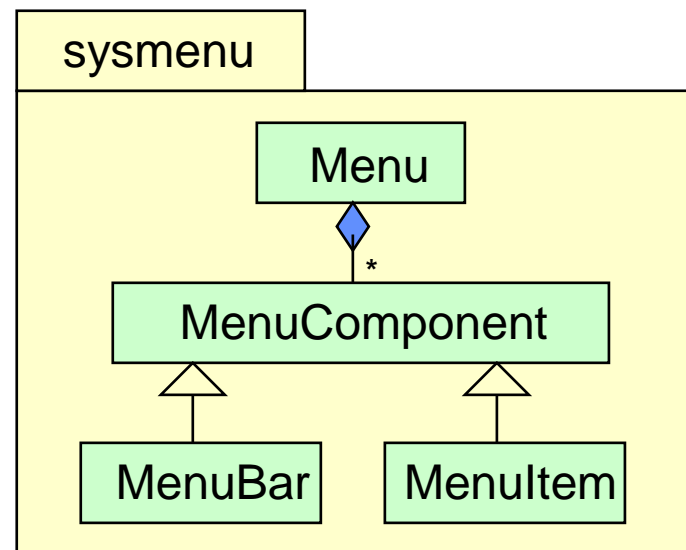
równoprawne (peer-to-peer)

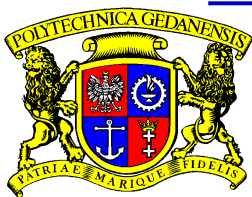




Pakiety UML

- Zestaw klas i związków pomiędzy klasami
- Odwzorowują zależności pomiędzy częściami systemu
- Mogą być zagnieżdżane
- Odwołanie do klasy zawartej w pakiecie: *nazwa-pakietu :: nazwa-klasy*
np. `sysmenu :: Menu`





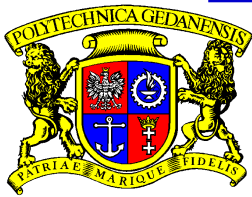
Pakiety UML



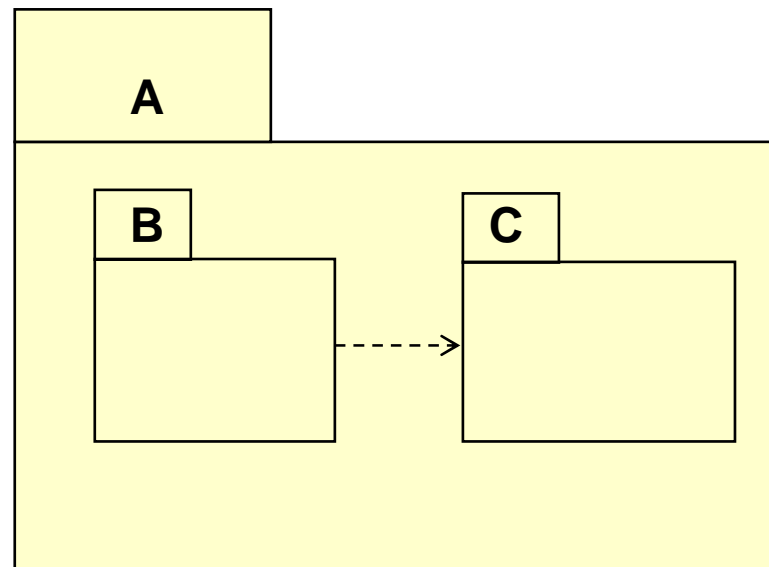
Pakiet A zależy od pakietu B (korzysta z jego funkcji)



Pakiety A i B korzystają wzajemnie ze swoich funkcji

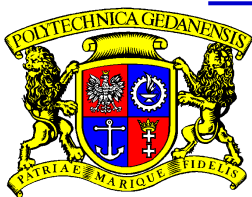


Pakiety UML



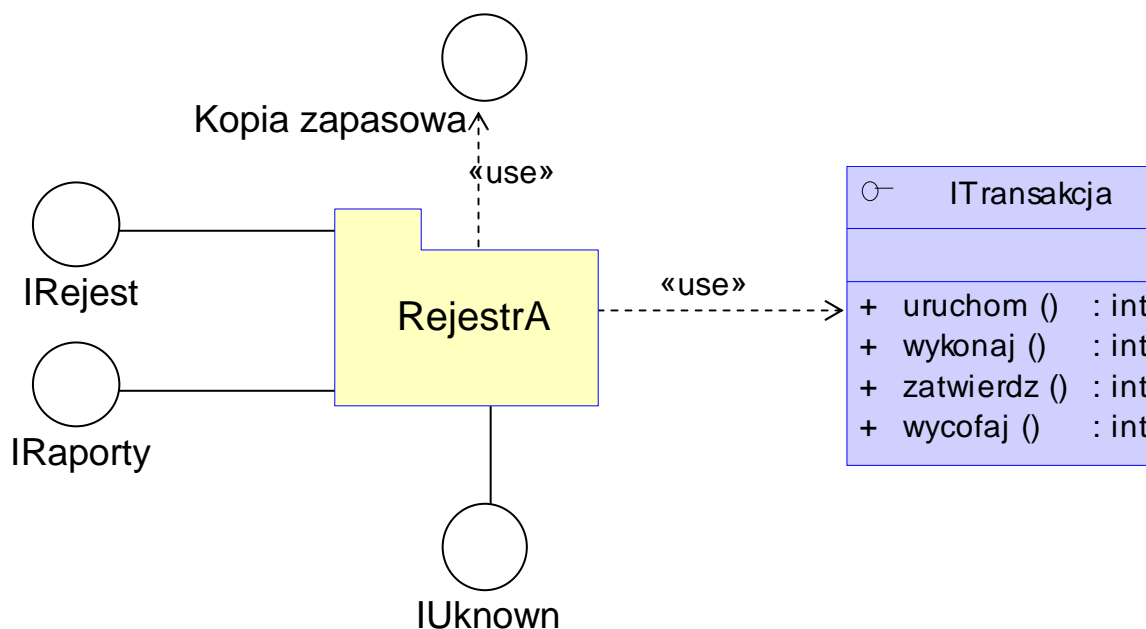
**Pakiety B i C są zagnieżdżone w pakiecie A.
Pakiet B zależy od pakietu C.**

Podsystemy zagnieżdżone (mniejsze) często nazywamy *modułami*.
Prosty system może składać się z kilku modułów (pakietów).

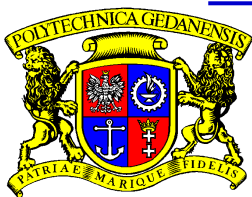


Podsystemy a interfejsy

Interfejsy stanowią wygodny sposób definiowania powiązań pomiędzy częściami systemu (podsystemami, modułami, ...), tzw. „szwów”.

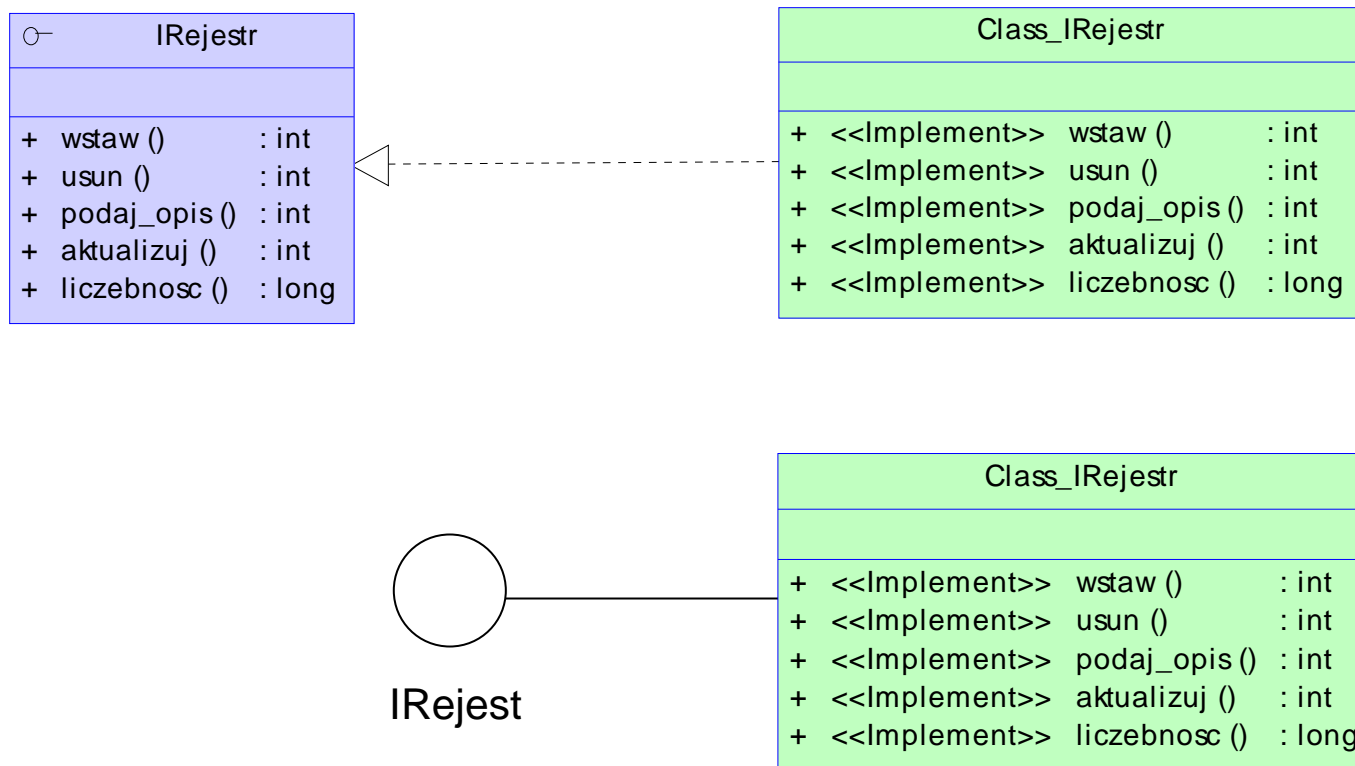


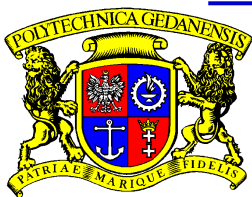
Zdefiniowanie interfejsów umożliwia rozdzielenie systemu na elementy składowe (komponenty) i następnie wykorzystywanie **dowolnych** komponentów, które działają zgodnie z tymi interfejsami.



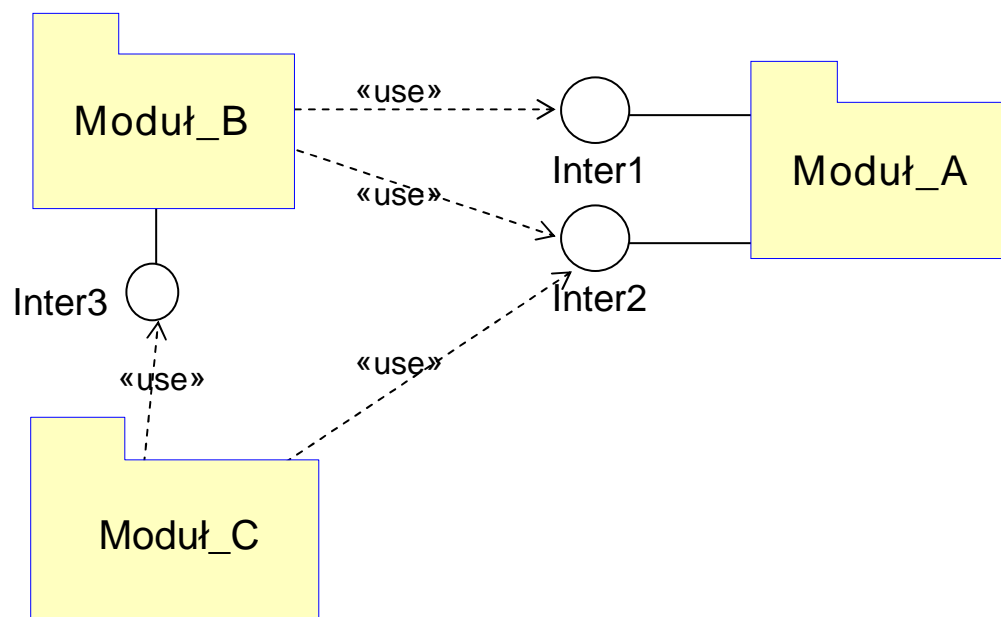
Interfejsy a klasy

Klasa może realizować jeden lub kilka interfejsów.





Podsystemy i interfejsy



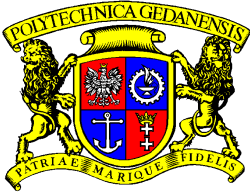
Moduł_A realizuje interfejsy **Inter1** i **Inter2**.

Moduł_B realizuje interfejs **Inter3**.

Moduł_B wykorzystuje interfejs **Inter1** i **Inter2**.

Moduł_C wykorzystuje interfejs **Inter2** i **Inter3**.

Wszystkie interfejsy są pokazane w formie skróconej, w związku z czym muszą być zdefiniowane w postaci rozszerzonej (kanonicznej) na innych diagramach modelu.



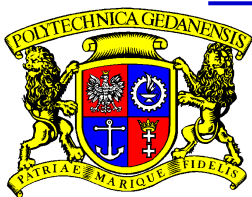
Identyfikacja współbieżności

- W modelu analitycznym (podobnie jak świecie rzeczywistym) wszystkie obiekty są współbieżne
- W implementacji - jeden procesor może obsługiwać wiele obiektów
- Analiza modeli dynamicznych pod kątem wymagań dotyczących współbieżności obiektów
 - wzajemna synchronizacja zdarzeń
 - czasowe zależności pomiędzy fizycznymi obiektami
 - identyfikacja wzajemnego wykluczania się operacji wykonywanych przez obiekty

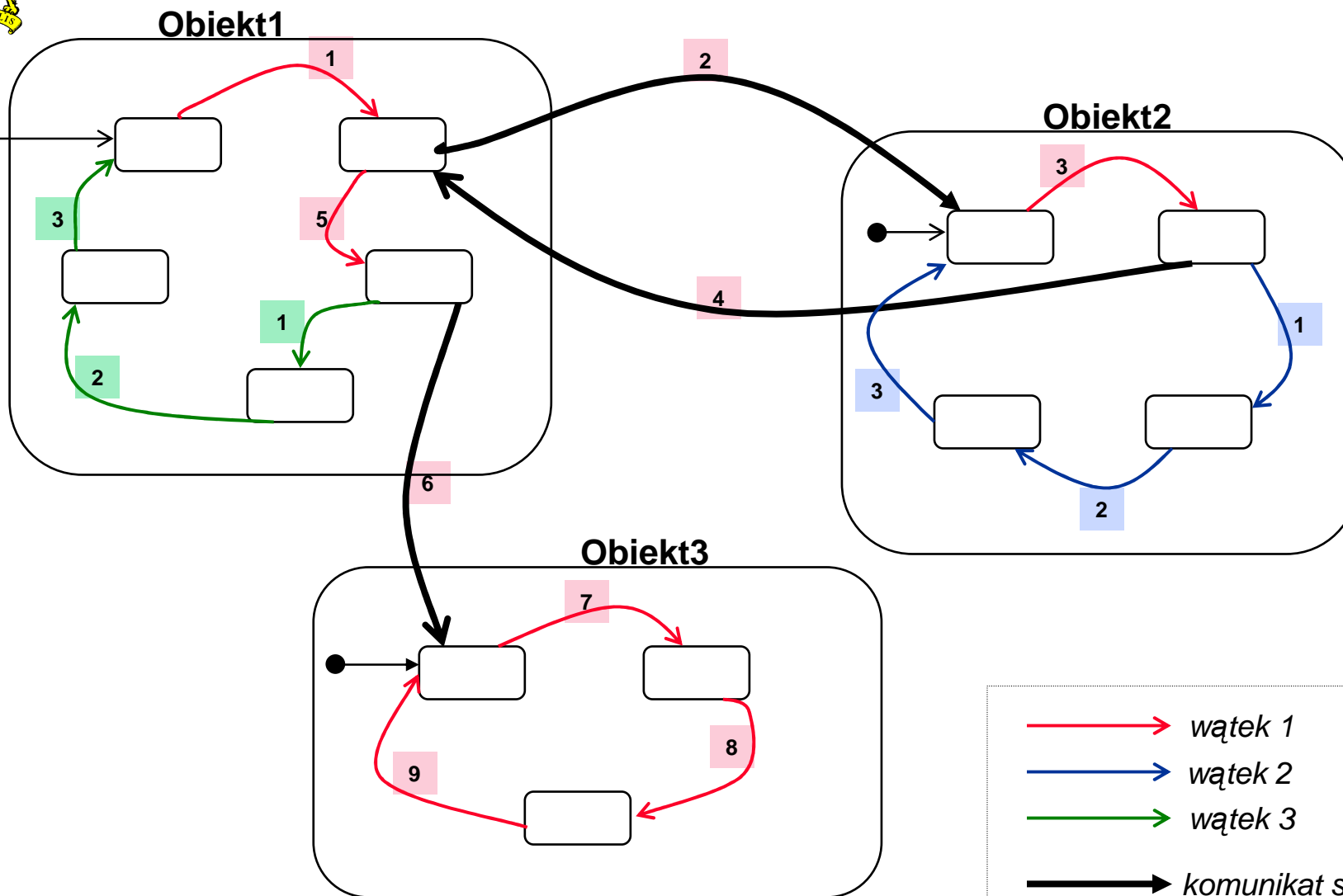


Wątek sterowania (*thread of control*) -

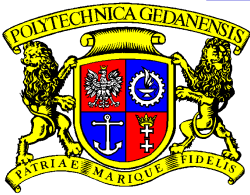
hipotetyczna ścieżka prowadzona przez diagramy stanów obiektów, wskazująca powiązania wynikające z przekazywania zdarzeń pomiędzy obiektami. W obrębie tej ścieżki zawsze jest aktywny tylko jeden z obiektów.



Wątki sterowania - przykład



- wątek 1
- wątek 2
- wątek 3
- komunikat synchr.
- komunikat asynchr.



Przydzielenie podsystemów do procesorów

Procesor – dowolne urządzenie fizyczne realizujące zadania systemu:
komputer, drukarka, czujnik, urządzenie specjalizowane, ...

Przesłanki przydzielania podsystemów do procesorów:

- wymagania dotyczące lokalizacji oprogramowania (np. sterowanie urządzeniem)
- redukcja transmisji danych pomiędzy komponentami
- ograniczenia zasobów sprzętowych
- wymagania efektywnościowe
- bezpieczeństwo

Kwestie do rozważenia:

- zwielokrotnienie procesorów (koszt a korzyści)
- obciążenie (średnie i szczytowe)
- akceptowany poziom błędów związanych z niedostępnością zasobów
- realizacja komunikacji pomiędzy procesorami:
topologia połączeń, kanały, protokoły, format przesyłanych danych, ...

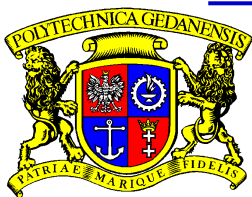


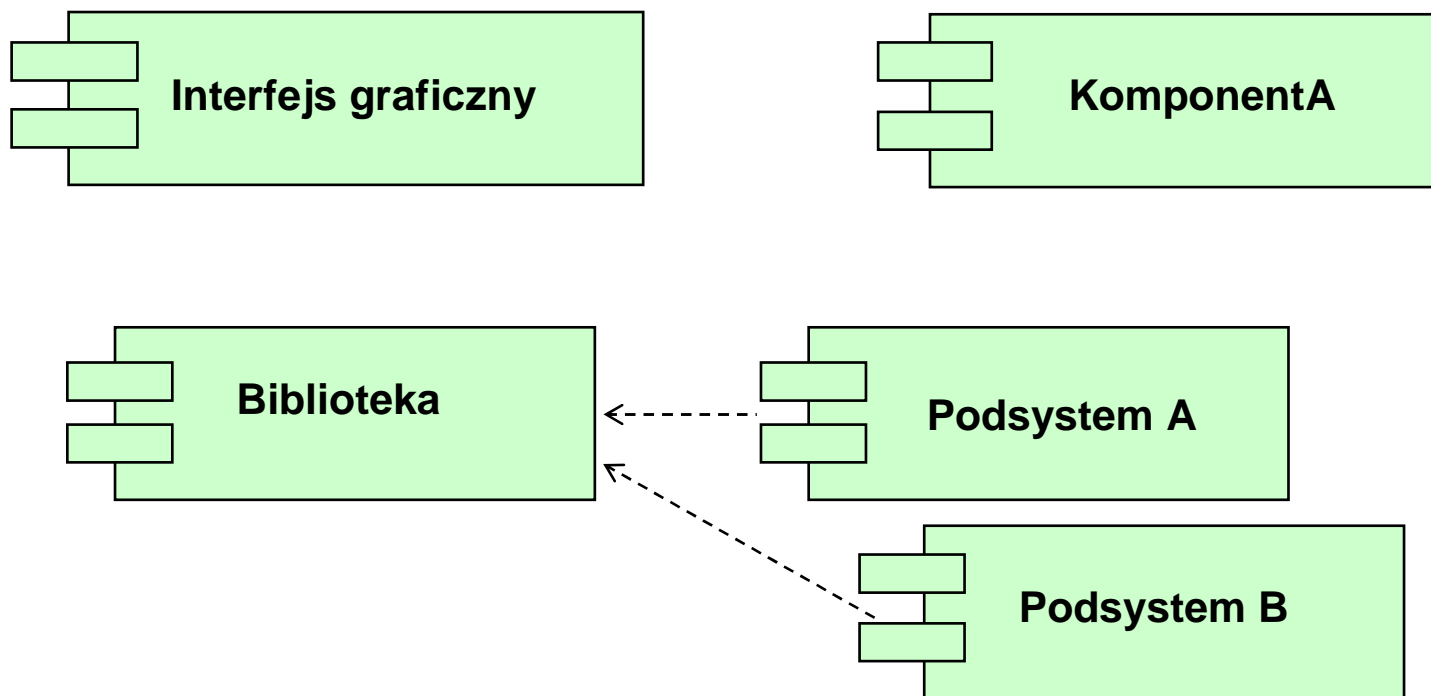
Diagram komponentów

Komponent - grupa obiektów realizująca zadanie

(komponenty kodu źródłowego, binarnego i wykonywalnego).

Strzałki pokazują zależności pomiędzy komponentami.

Diagram implementacyjny:
pokazuje strukturę konstrukcyjną kodu systemu.



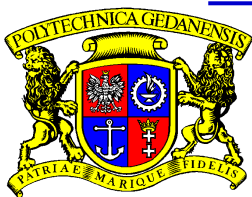


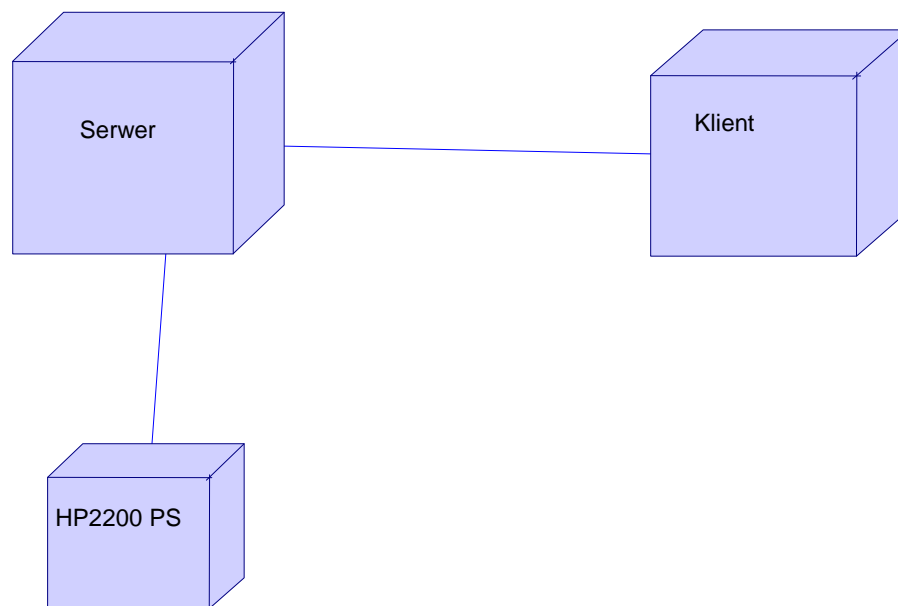
Diagram rozmieszczenia

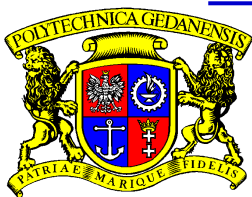
Opis fizycznej topologii systemu:

Węzłami są procesory połączone przez linie odwzorowujące komunikację pomiędzy nimi.

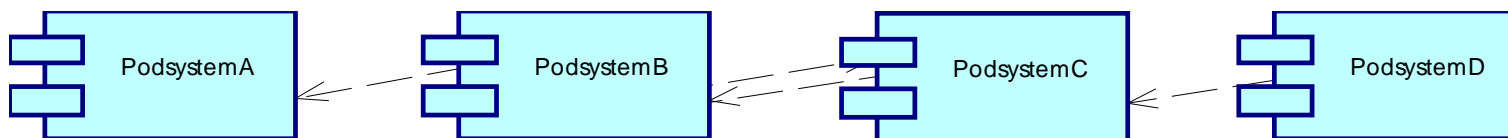
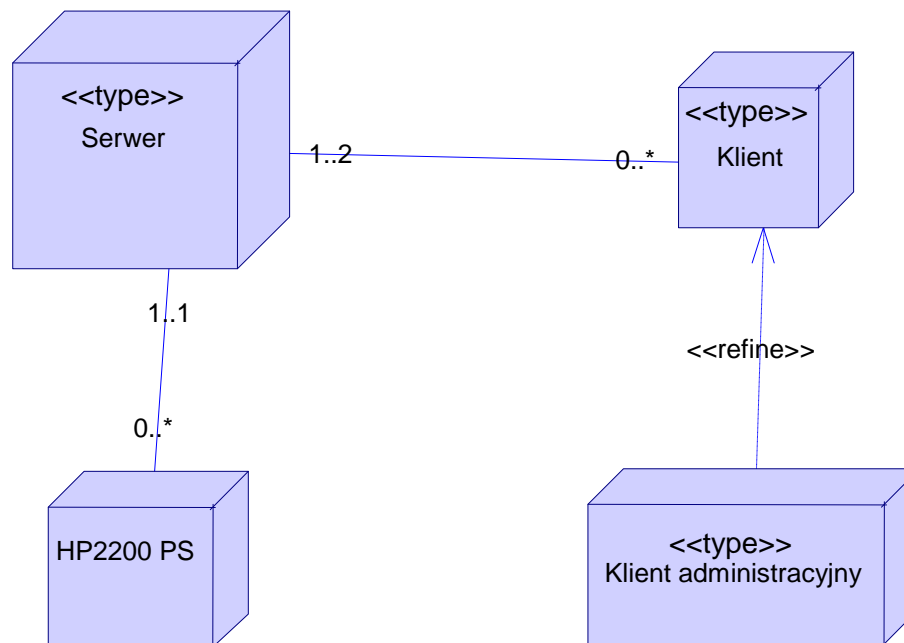
Przedstawia architekturę procesorów i aktywnych komponentów oprogramowania.

Diagram implementacyjny:
pokazuje strukturę systemu w czasie jego funkcjonowania w środowisku docelowym.





Diagramy rozmieszczenia i komponentów



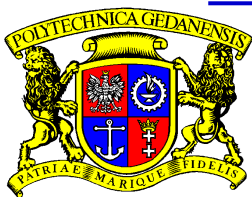
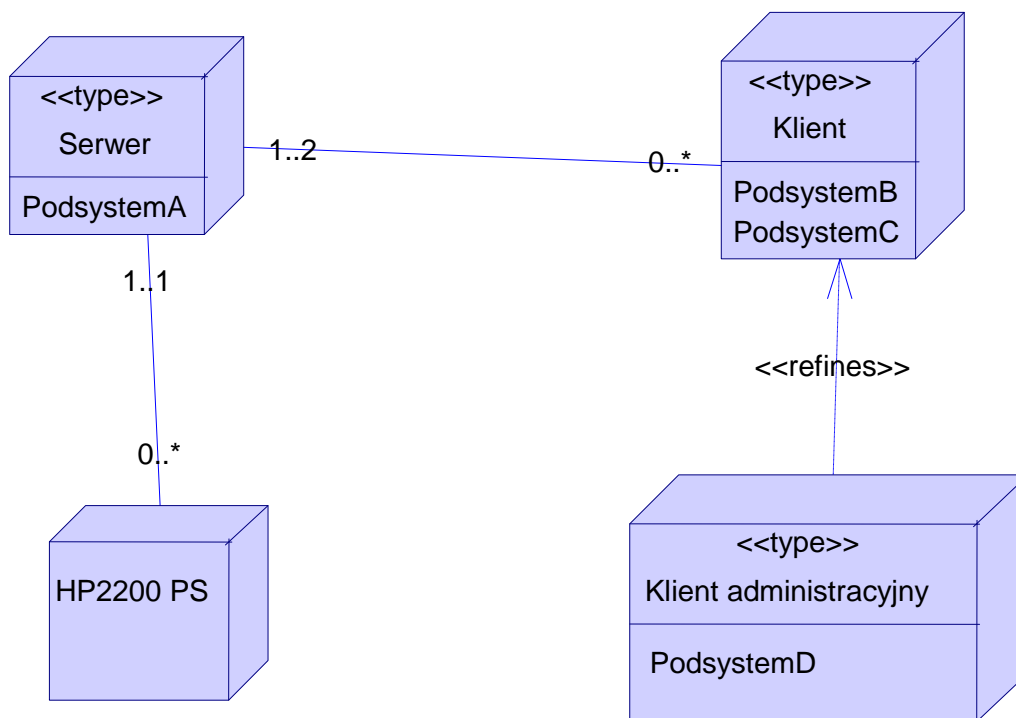
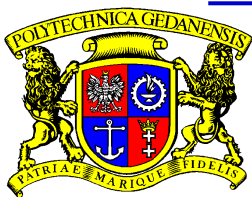


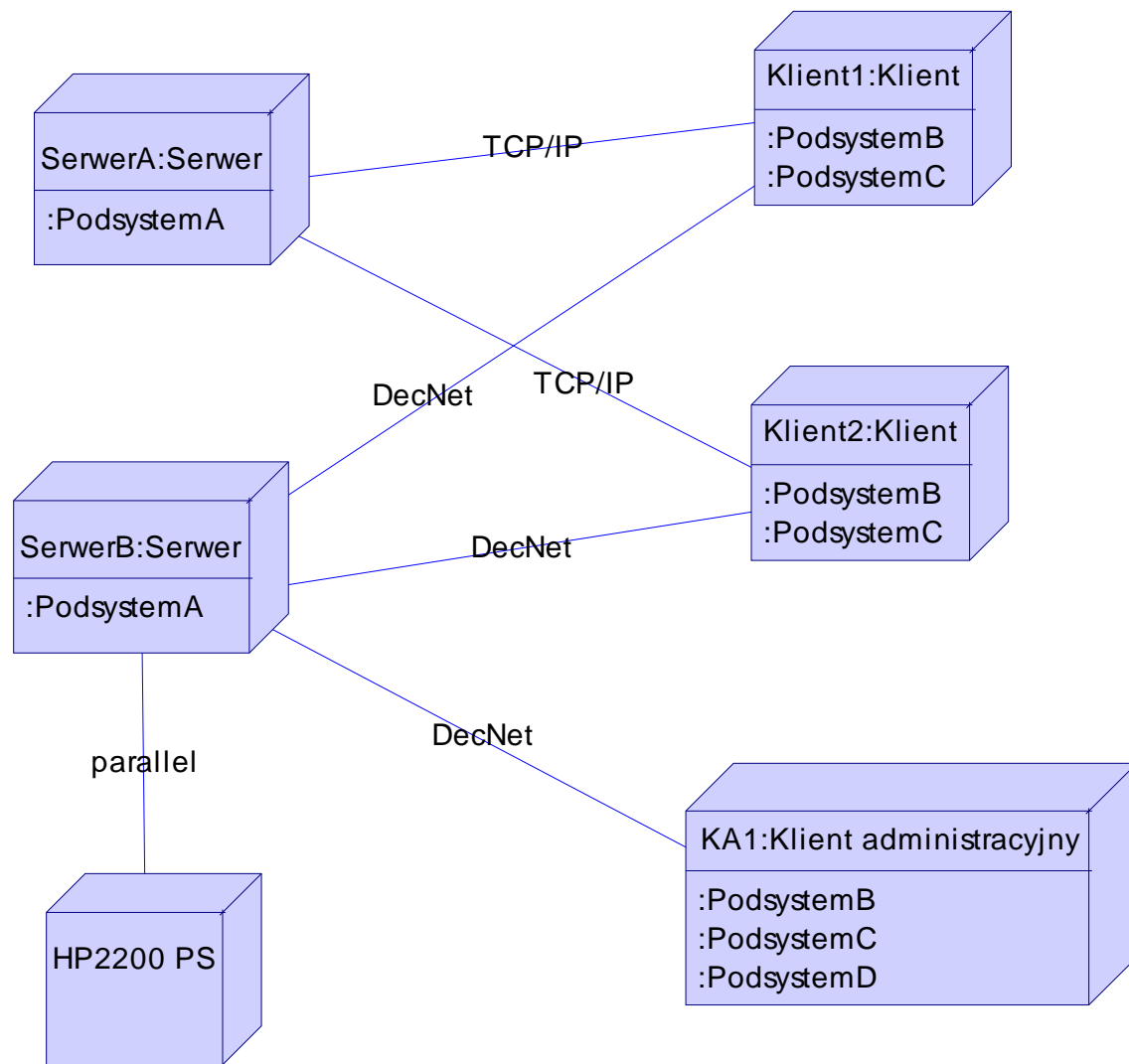
Diagram rozmieszczenia – węzły z komponentami

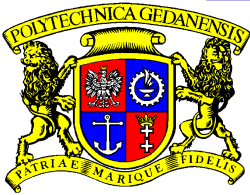


Komponenty zostały przypisane do węzłów, na których będą rezydować.



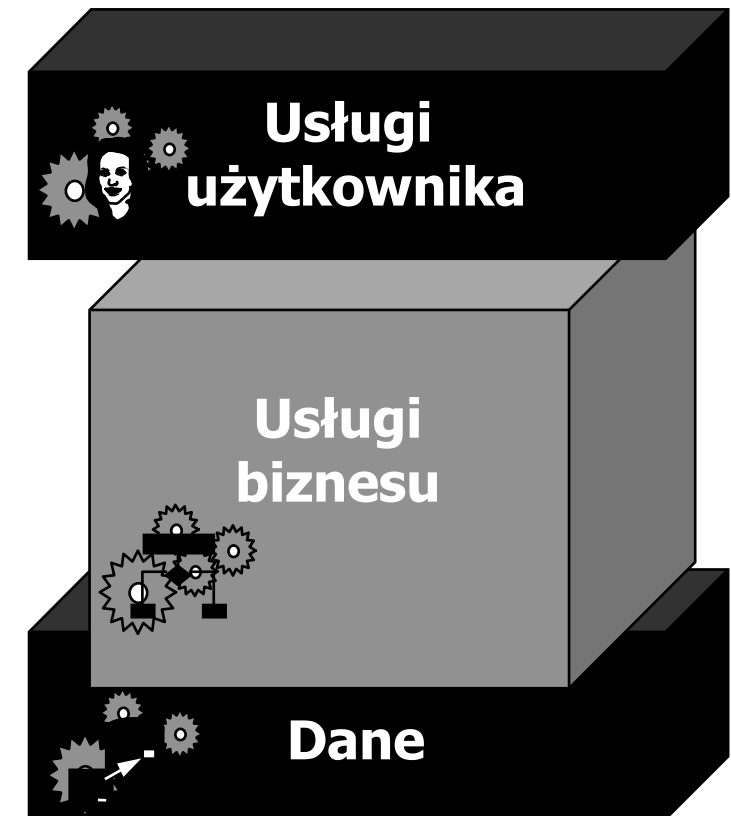
Instancja diagramu rozmieszczenia

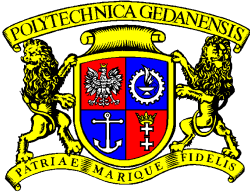




Architektura trójwarstwowa

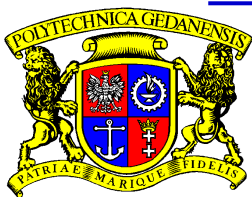
- **Obiekty interfejsu**
 - realizują styk użytkownik – oprogramowanie
 - cienki interfejs – nie realizuje żadnych usług – dostarcza informacje do warstwy sterującej
 - inteligentny interfejs – potrafi wykonać określone funkcje biznesowe
- **Usługi biznesu – obiekty sterujące**
 - realizują przypadki użycia
 - decydują o sekwencji wywołań usług warstwy sterującej
- **Usługi biznesu – obiekty biznesowe**
 - realizują usługi aplikacji
 - obiekty trwałe (persistent) posiadają metody ułatwiające ich odczytanie z bazy danych
- **Dostęp do danych**
 - sprzęg z relacyjną bazą danych



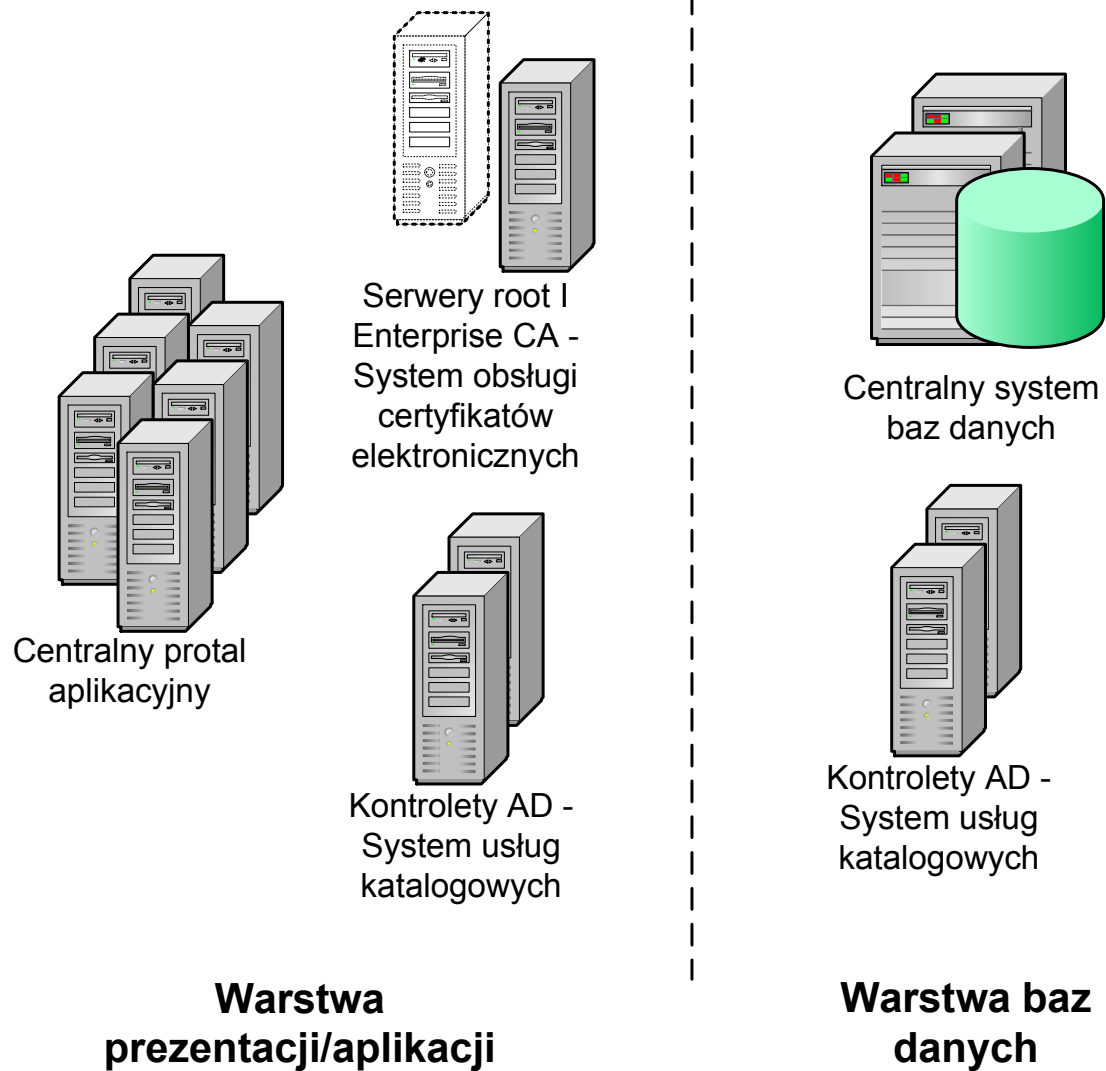


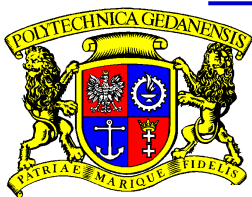
Projekt infrastruktury techniczno-systemowej (ITS)

- **Założenia, wymagania oraz specyfikacje dla elementów infrastruktury techniczno-systemowej (ITS)**
 - **Założenia i wymagania dla infrastruktury teleinformatycznej – założenia dla środowiska eksploatacyjnego, oszacowania wolumenu danych pojemników danych, sieci, parametry dostępowe, Service Level Agreement (SLA), zasady monitorowania zdarzeń, systemy kopii zapasowych, standardy stacji roboczych itp.**
 - **Specyfikacja elementów ITS**
 - **Specyfikacja techniczna serwerów**
 - **Specyfikacja techniczna sieciowych urządzeń aktywnych**
 - **Specyfikacja techniczna systemów pamięci masowych**

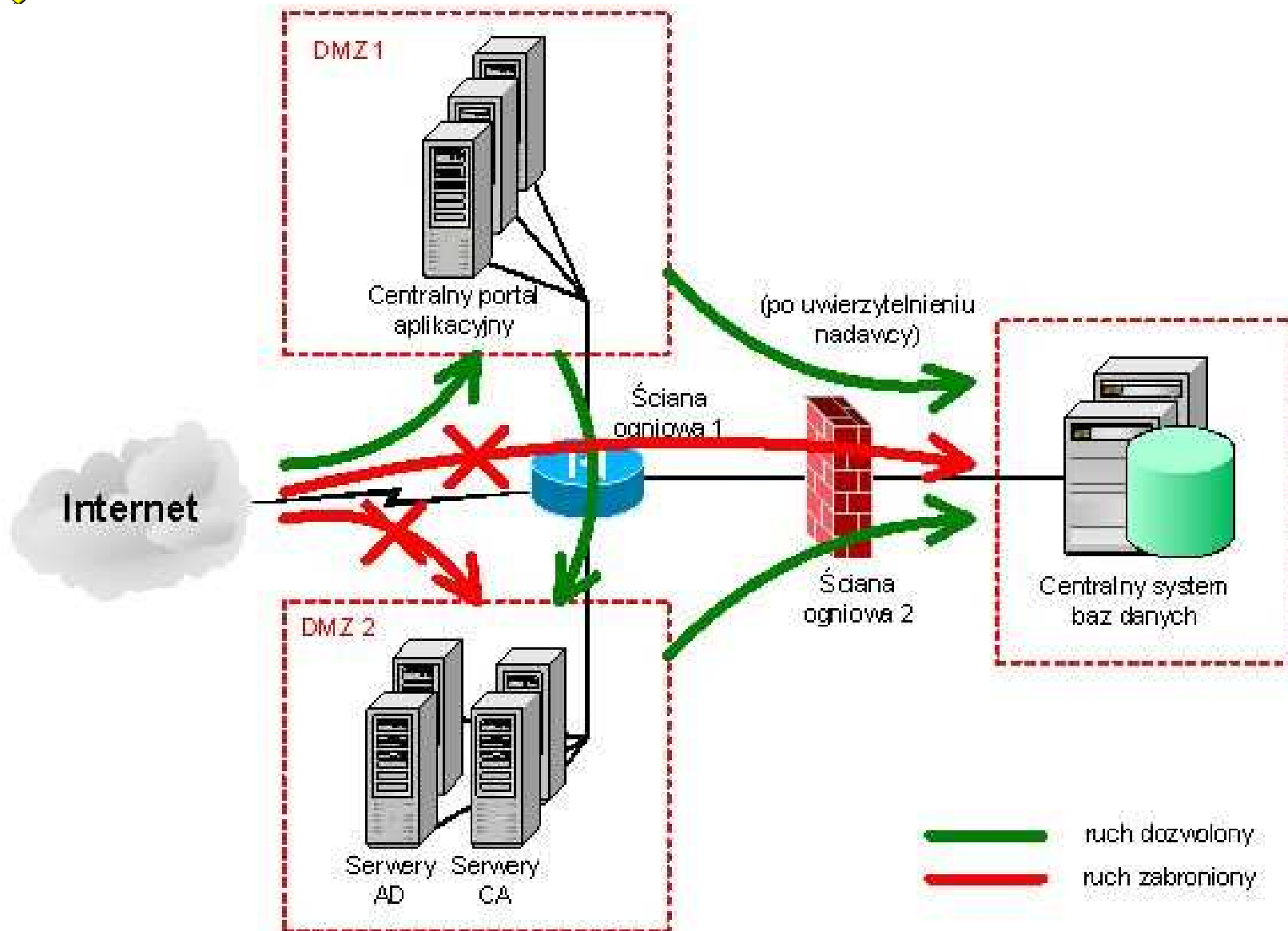


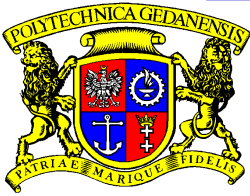
Projekt ITS – podział na warstwy





Projekt ITS – infrastruktura sieciowa



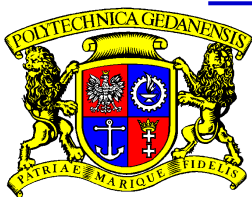


Zarządzanie pojemnikami danych

Pojemniki danych służą jako zasoby umożliwiające przechowywanie i współużytkowanie informacji. Mogą także służyć jako interfejs pomiędzy podsystemami lub innymi elementami systemu.

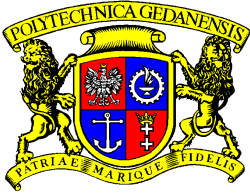
- Sposoby realizacji pojemników danych:
 - pliki
 - bazy danych
 - struktury w pamięci operacyjnej)
- Wybór rozwiązania musi uwzględniać:
 - koszt
 - czas dostępu
 - pojemność
 - niezawodność

Projekt logiczny i fizyczny bazy danych powstaje na etapie projektowania klas, kiedy to decyduje się o trwałości obiektów.



Zarządzanie pojemnikami danych

P L I K I	B A Z Y D A N Y C H
<p>(+) niski koszt (+) prostota obsługi</p>	<p>(+) zaawansowane funkcje obsługi danych (współbieżność, transakcje, archiwizacja,...) (+) model danych (+) standardowy język dostępu</p>
<p>(-) brak struktury informacyjnej danych (-) brak zaawansowanych funkcji obsługi</p>	<p>(-) wysoki koszt (-) duże wymagania sprzętowo-programowe (-) narzuty na wydajność</p>
<ul style="list-style-type: none">• Obszerne wolumeny danych, pozbawione wyraźnej struktury• Dane archiwalne, kopie, ślady wykonania, logi, ...• Dane tymczasowe• Masowe dane będące źródłem informacji dla baz danych	<ul style="list-style-type: none">• Dane dostępne dla wielu użytkowników, z różnych perspektyw• Dane dostępne z różnych systemów (platform)• Dane współużytkowane przez wiele programów użytkowych (użytkowników)• Dane wymagające specjalnej ochrony



Obsługa zasobów globalnych

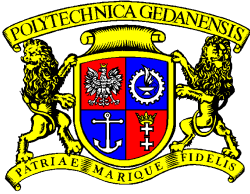
Zasób globalny – pasywny element systemu dostępny (zazwyczaj współbieżnie) dla wielu obiektów

Zasoby fizyczne:

- jednostka fizyczna
(procesor, dysk, urządzenie komunikacyjne, czujnik, ...)
- przestrzeń działania
(obszar w pamięci operacyjnej, miejsce na dysku, ekran stacji roboczej)
- ...

Zasoby logiczne:

- nazwy logiczne (identyfikatory, nazwy plików, nazwy obiektów, ...)
- dane (w bazie danych, w pamięci operacyjnej)
- ...



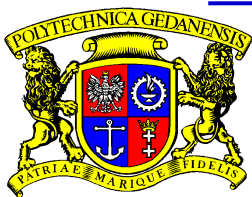
Obsługa zasobów globalnych

Problem:

synchronizowanie dostępu do zasobu globalnego
w środowisku współbieżnym (zasobu współużytkowanego)

Metody:

- Zastosowanie obiektu synchronizującego dostęp (*guardian object*) - obiekt taki działa we własnym wątku, a wszystkie operacje dostępu do zasobu są realizowane za jego pośrednictwem. Metoda bezpieczna (pod warunkiem jej rygorystycznego stosowania!), lecz nieefektywna czasowo.
- Wykorzystanie mechanizmów synchronizacyjnych systemu operacyjnego: semaforów, sygnałów, ... Metoda niebezpieczna (uwaga – deadlock!), lecz efektywna czasowo.
- Wykorzystanie mechanizmów systemu zarządzania bazami danych. Metoda właściwa do synchronizacji dostępu do danych.



Obsługa warunków granicznych

- **INICJALIZACJA**

Przejście od stanu spoczynku do ustabilizowanego stanu pracy

- inicjowanie danych stałych, parametrów, zadań, kanałów komunikacyjnych,...
- sytuacje szczególne przy pierwszym uruchomieniu systemu



- **ZAKOŃCZENIE**

Przejście od stanu pracy do stanu spoczynku

- zwolnienie zasobów
- likwidacja obiektów nietrwałych
- systematyczne zakończenie zadań zależnych od siebie

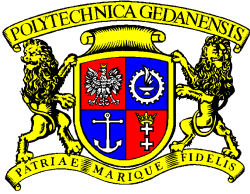


- **UPADEK**

Nieplanowane zakończenie pracy systemu

- zarejestrowanie przyczyn i stanu systemu (jeśli możliwe)





Wybór priorytetów

Priorytety - często są trudne do zmierzenia i nie dają się wzajemnie porównać.

Bierze się pod uwagę nie tylko docelowy system, ale i proces jego budowy, np.:

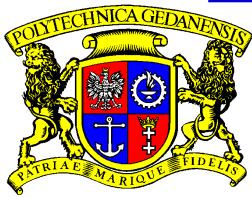
- Termin instalacji ważniejszy niż wydajność, bo grożą kary umowne.
- Zadowolenie klienta jest bardzo ważne, ale firma nie może zbankrutować!
- Projektujemy z myślą o *reuse*, ale terminy gonią...

- **Przykładowe priorytety:**

- zapotrzebowanie na zasoby sprzętowe
- zapotrzebowanie na pamięć operacyjną i zewnętrzną
- wydajność czasowa
- łatwość i koszt utrzymania
- przenośność
- odporność na awarie
- kompletność i przejrzystość projektu
- modyfikowalność
- ...



Konieczne jest wypracowanie rozwiązań kompromisowych!



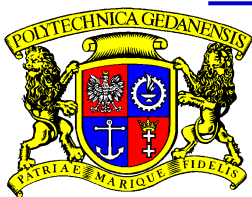
Projekt interfejsów systemu

1. Decyzje odnośnie:

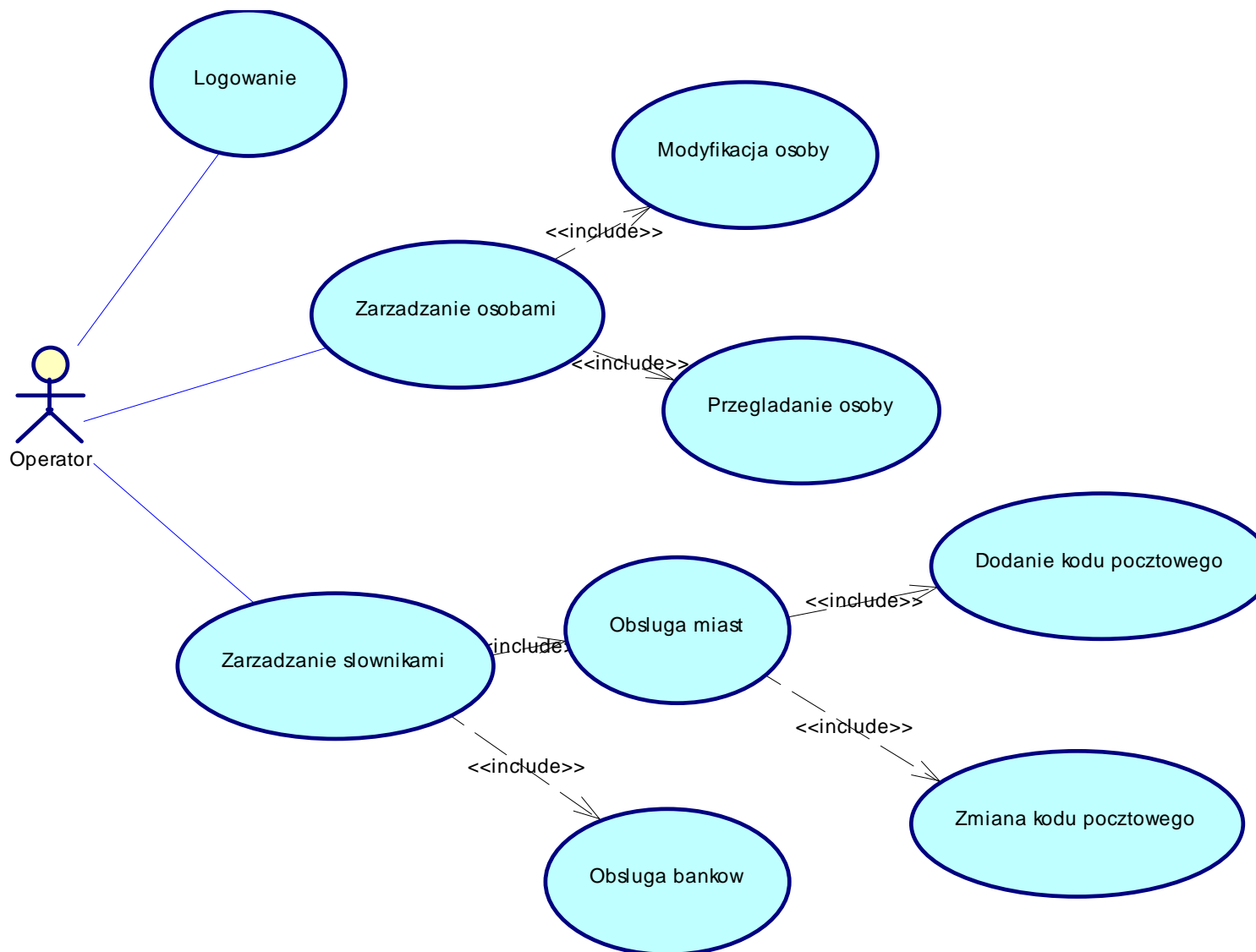
- rodzaju interfejsu użytkownika (tekstowy, graficzny, multimedialny, ...)
- zastosowania urządzeń specjalizowanych (np. do uwierzytelniania)
- potrzeby użycia/konstrukcji interfejsów do urządzeń zewnętrznych (np. czujników)
- realizacji interfejsu z innymi systemami

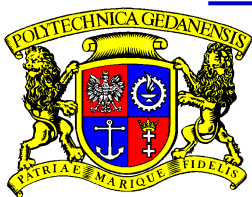
2. Projekt menu użytkownika:

- sposoby wywoływania funkcji systemu przez użytkownika
- ogólna struktura („mapa”) menu systemu

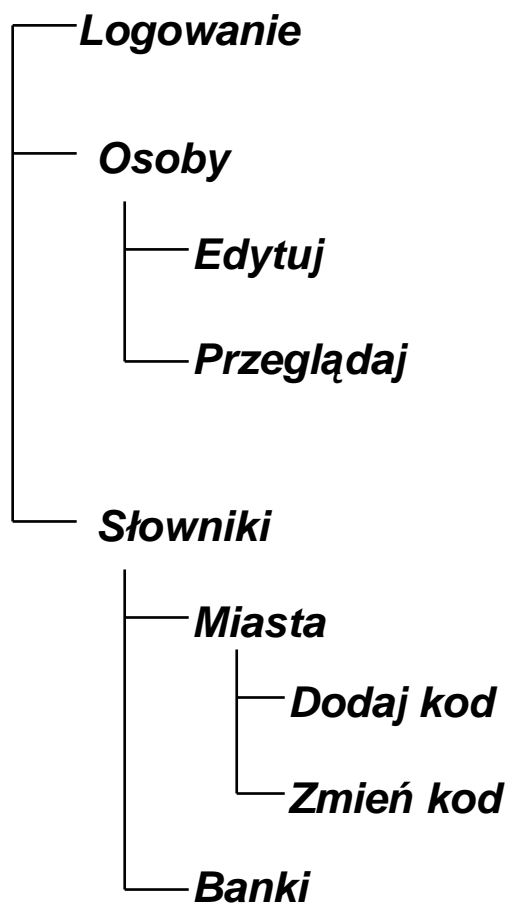


Przypadki użycia a menu systemu



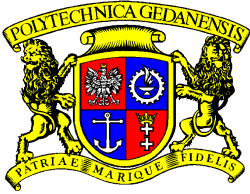


Przypadki użycia a menu systemu



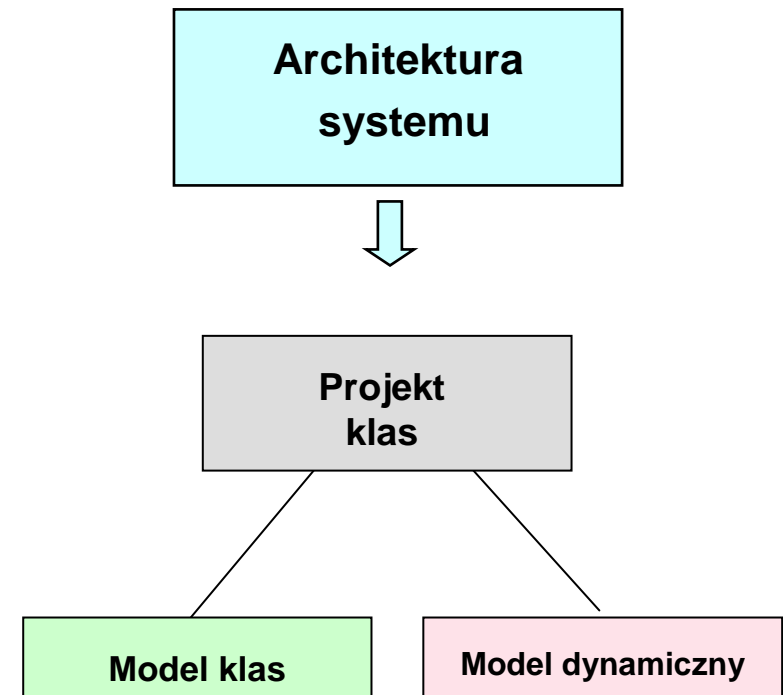
+

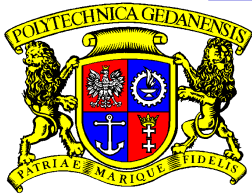
Opis słowny



Projektowanie klas

- **Modularyzacja klas**
- **Poprawienie struktury klas pod względem dziedziczenia**
- **Określenie reprezentacji klas**
- **Przydzielenie klas do warstw**
- **Zdefiniowanie schematu bazy danych**
- **Opracowanie algorytmów implementujących operacje**
- **Określenie implementacji związków**
- **Optymalizacja projektu**



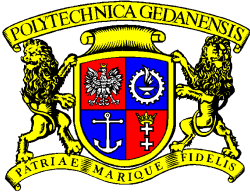


Aspekty projektowania klas

- Przesunięcie nacisku z pojęć dziedziny aplikacyjnej w kierunku pojęć informatycznych
- Modele z analizy łącznie z wynikami projektowania systemu są początkiem projektu szczegółowego
- Gromadzenie informacji w modelu klas - pełna definicja klas i związków przeznaczonych do implementacji
- Dodanie detali i podjęcie decyzji implementacyjnych
- Wyrażenie operacji algorytmami implementacyjnymi
- Optymalizacja danych i algorytmów: czas, pamięć, ...
- Kompromis między optymalizacją a czytelnością, łatwością implementacji, pielęgnacji, ponownej używalności, ...
- Odwzorowanie logicznej struktury analizy w fizyczną organizację programu



Wpływ środowiska implementacyjnego i środowiska pracy



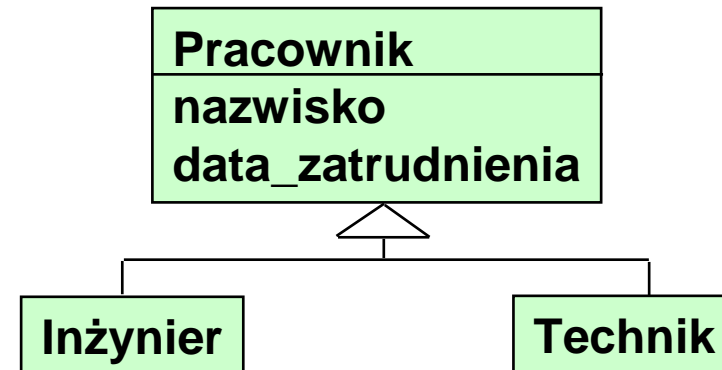
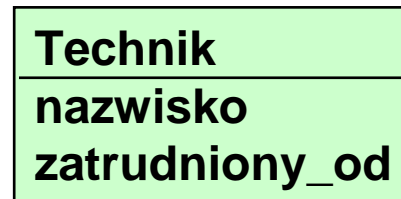
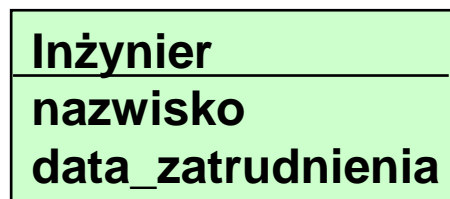
Modularyzacja klas

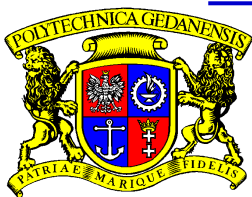
- **Spójność semantyczna: klasa dotyczy jednego aspektu aplikacji**
- **Klasa realizuje tylko kilka zadań**
 - podział klasy: generalizacja, agregacja
 - nie więcej niż 20 atrybutów, 10 asocjacji i 20 operacji
- **Metoda realizuje tylko jedno zadanie**
- **Techniki polepszające modularność dotyczące atrybutów**
 - atrybuty prywatne (hermetyzacja)
 - dostęp do atrybutów przez operacje



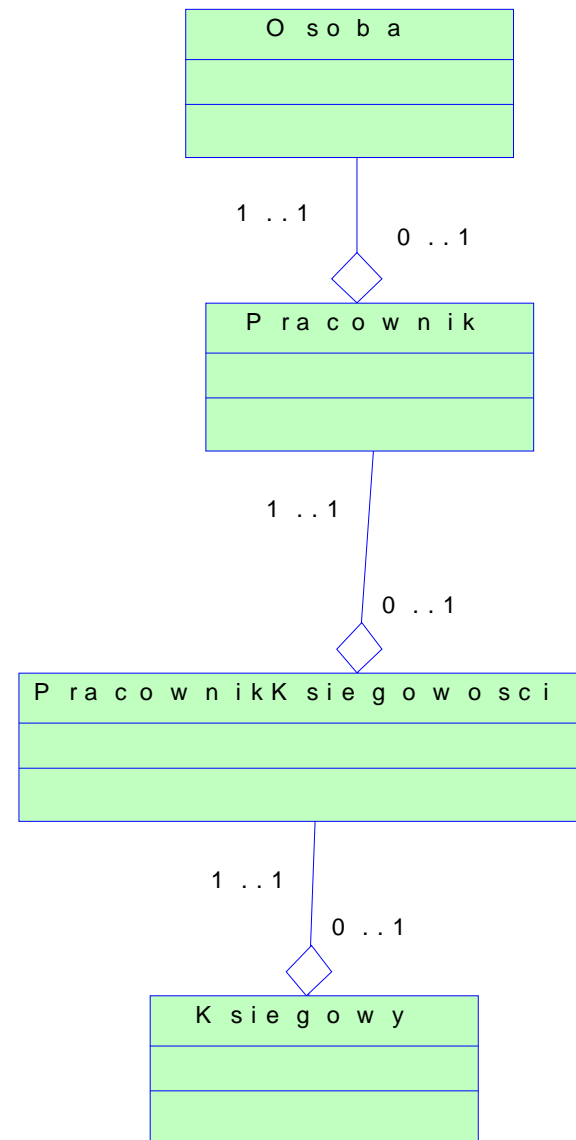
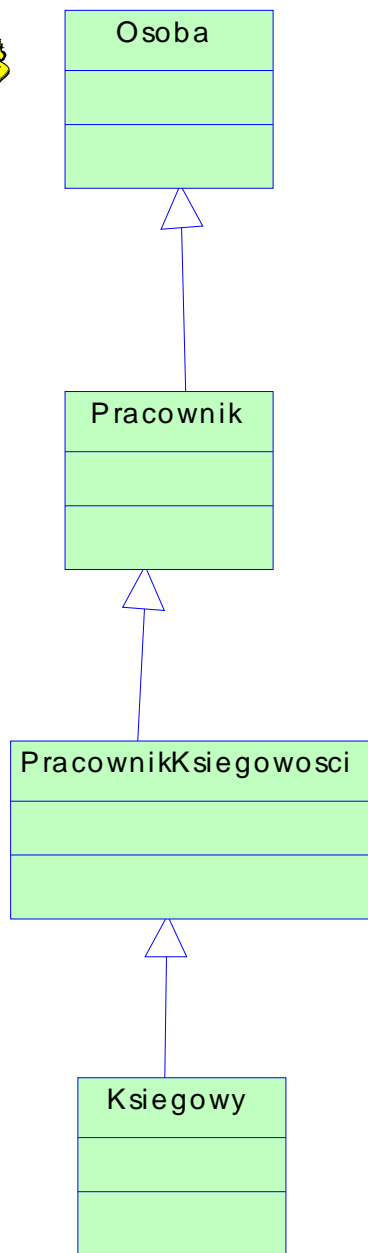
Systematyzacja dziedziczenia

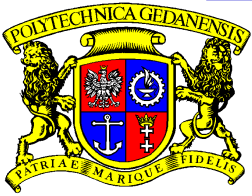
- **Dziedziczenie polepsza**
 - współdzielenie danych i kodu
 - modularność: rozdzielenie części wspólnych i specyficznych
 - ponowną używalność
 - rozszerzalność
 - zarządzanie konfiguracją
- **Wyszukiwanie wspólnej struktury i wspólnego zachowania**
 - definiowanie wspólnej nadklasy (abstrakcyjnej) dla podobnych klas
 - jeżeli podobne znaczeniowo atrybuty mają inne nazwy, należy ustalić wspólną nazwę i przesunąć atrybut do nadklasy





Dziedziczenie a agregacja

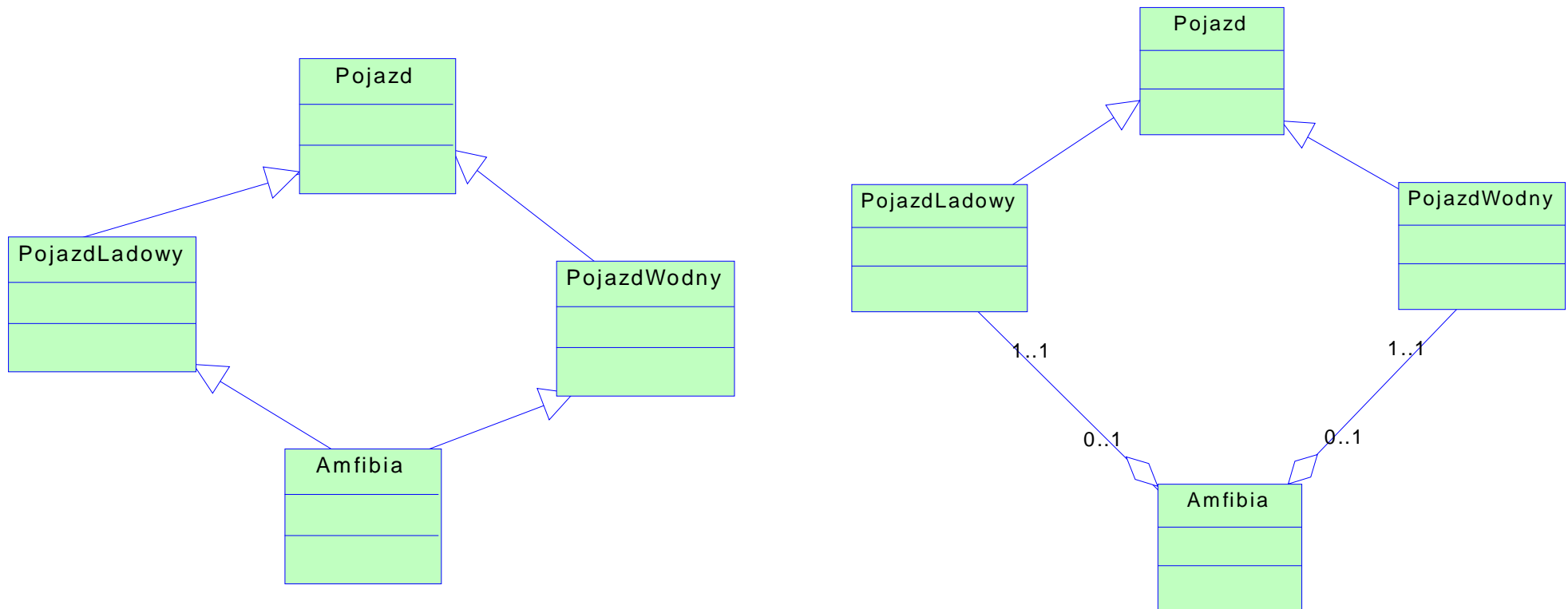


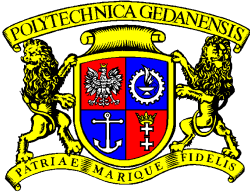


Systematyzacja dziedziczenia

- **Wielodziedziczenie**

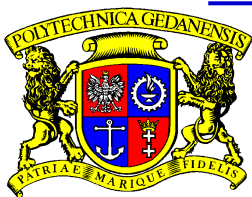
- czy język programowania dopuszcza wielodziedziczenie?
- zamiana wielodziedziczenia na agregację



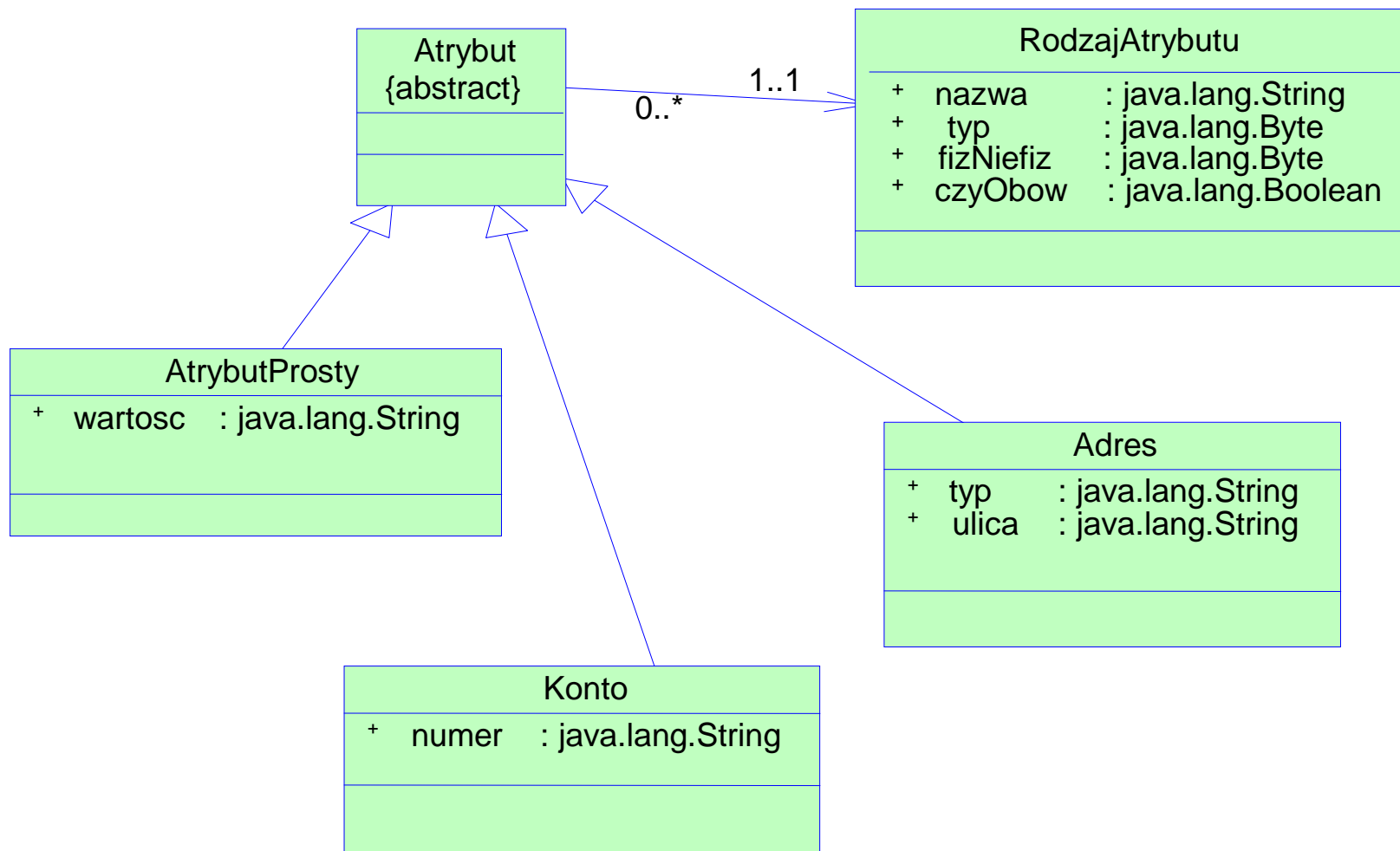


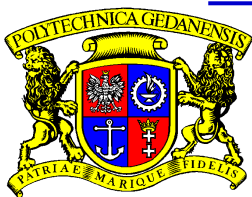
Systematyzacja dziedziczenia

- **„Spłaszczanie” dziedziczenia**
 - czy wszystkie klasy z hierarchii dziedziczenia mają atrybuty?
 - czy ich istnienie jest uzasadnione?
 - czy zachować klasy abstrakcyjne?
 - czy hierarchia dziedziczenia nie jest zbyt głęboka?
 - jaki jest koszt zachowania hierarchii dziedziczenia z etapu analizy?

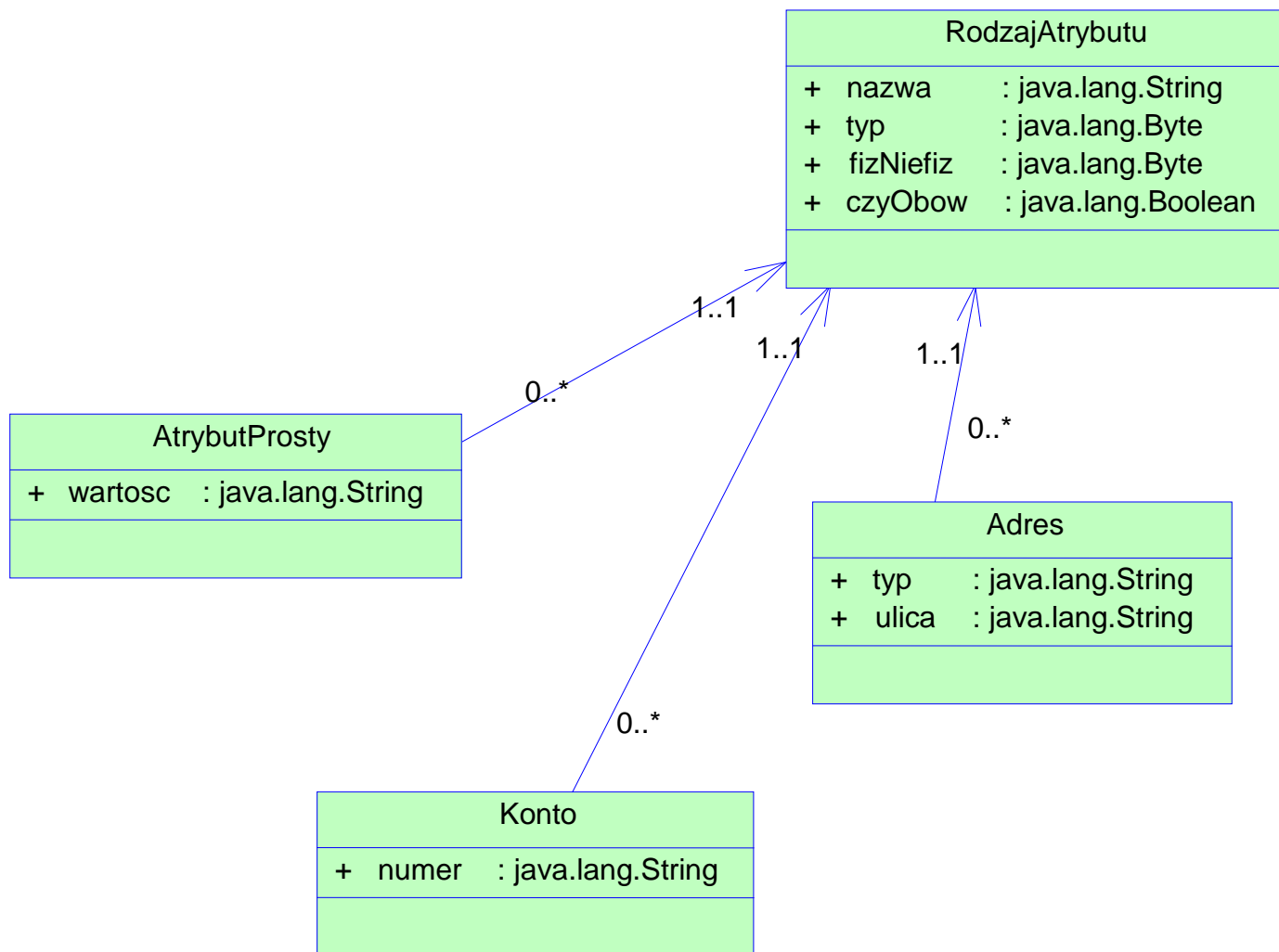


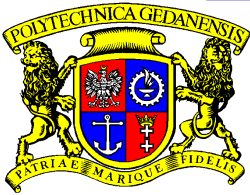
Splaszczanie dziedziczenia (1)





Splaszczanie dziedziczenia (2)

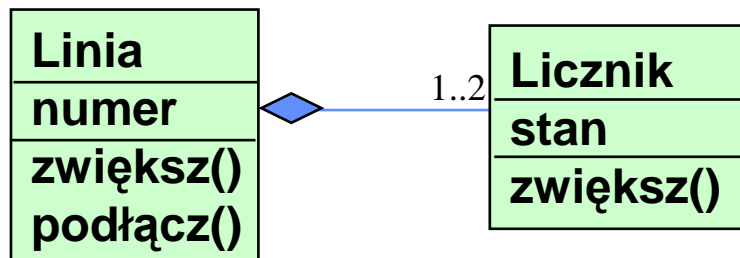




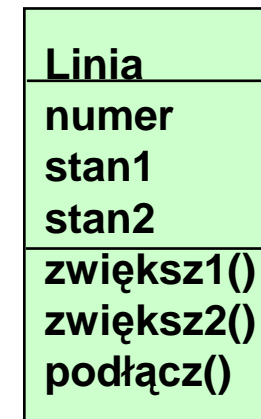
Określenie reprezentacji klas

- Klasy mogą być osadzane w innych klasach (implementowane jako atrybuty i operacje w innych klasach).
- Atrybut może być implementowany jako:
 - związek lub agregacja do innej klasy
 - wartość predefiniowanego typu

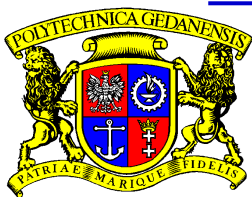
lepszą modyfikowalność



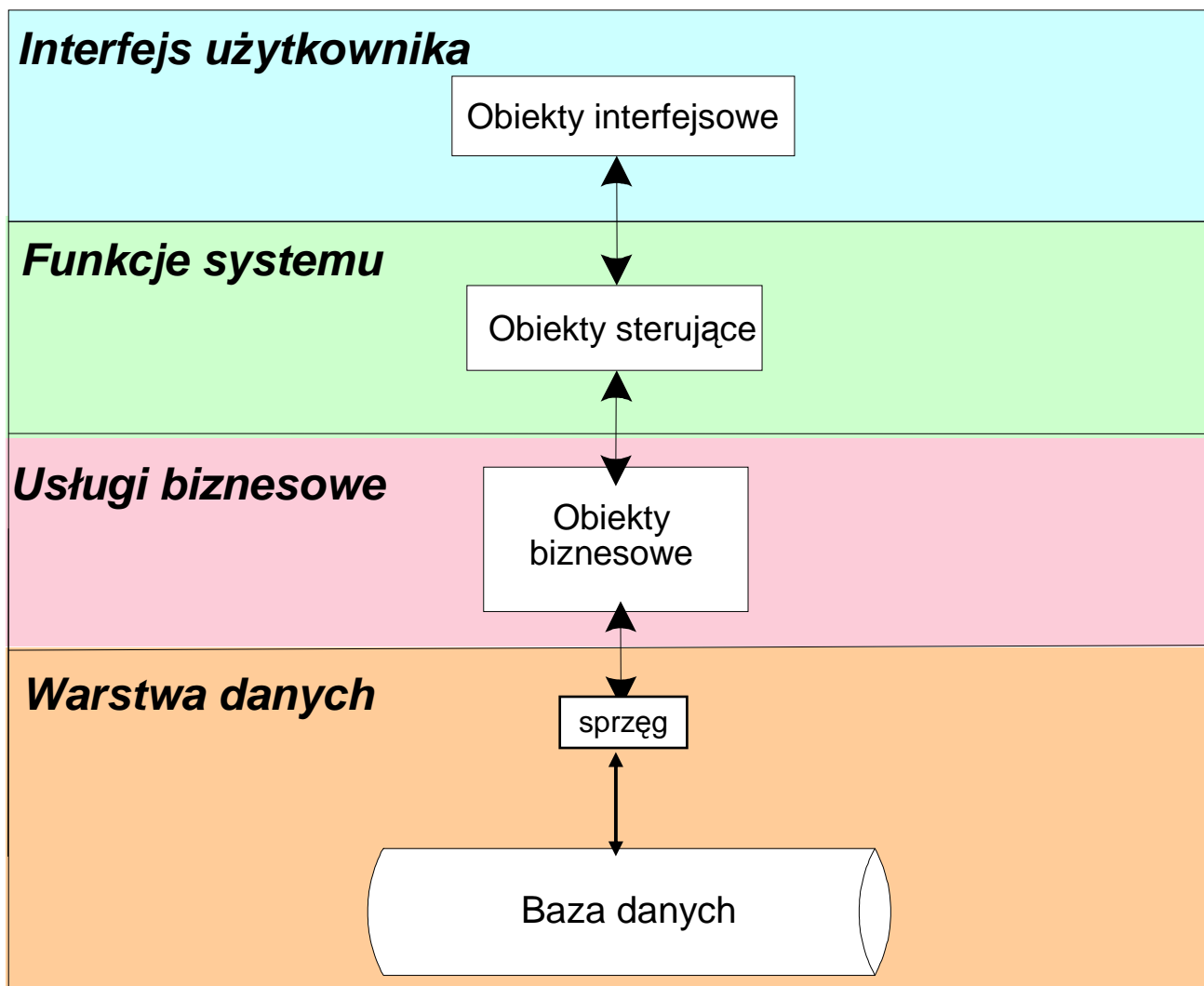
lepszą efektywność

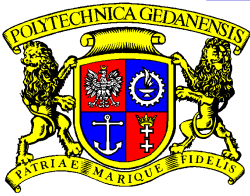


Osadzenie klasy Licznik w klasie Linia



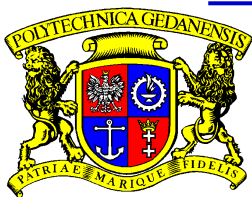
Przydzielenie klas do warstw





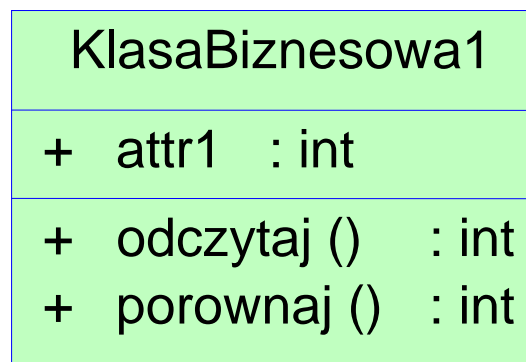
Rola obiektów w warstwach

- **Obiekty interfejsowe**
 - realizują styk użytkownik – oprogramowanie
 - może być to tzw. *cienki interfejs*, tzn. taki, który nie realizuje żadnych usług – dostarcza tylko informacji do warstwy sterującej
- **Obiekty sterujące**
 - realizują przypadki użycia (zwykle: jeden obiekt – jeden przypadek)
 - decydują o sekwencji wywołań usług warstwy biznesowej
- **Obiekty biznesowe**
 - realizują usługi aplikacji
 - mogą być obiektami trwałymi (*persistent*), mającymi metody umożliwiające ich zapis do bazy danych i ich odczyt z bazy danych
- **Warstwa danych**
 - sprzęg z bazą danych (np. gotowe komponenty środowiska implementacyjnego) i baza danych



Przykład architektury trójwarstwowej

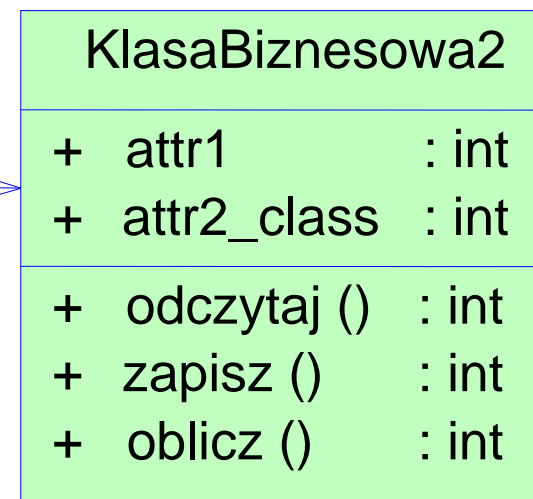
W modelu analitycznym nie ma jeszcze warstw:

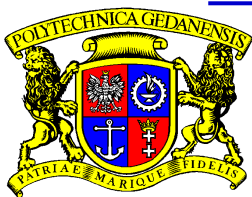


0..*

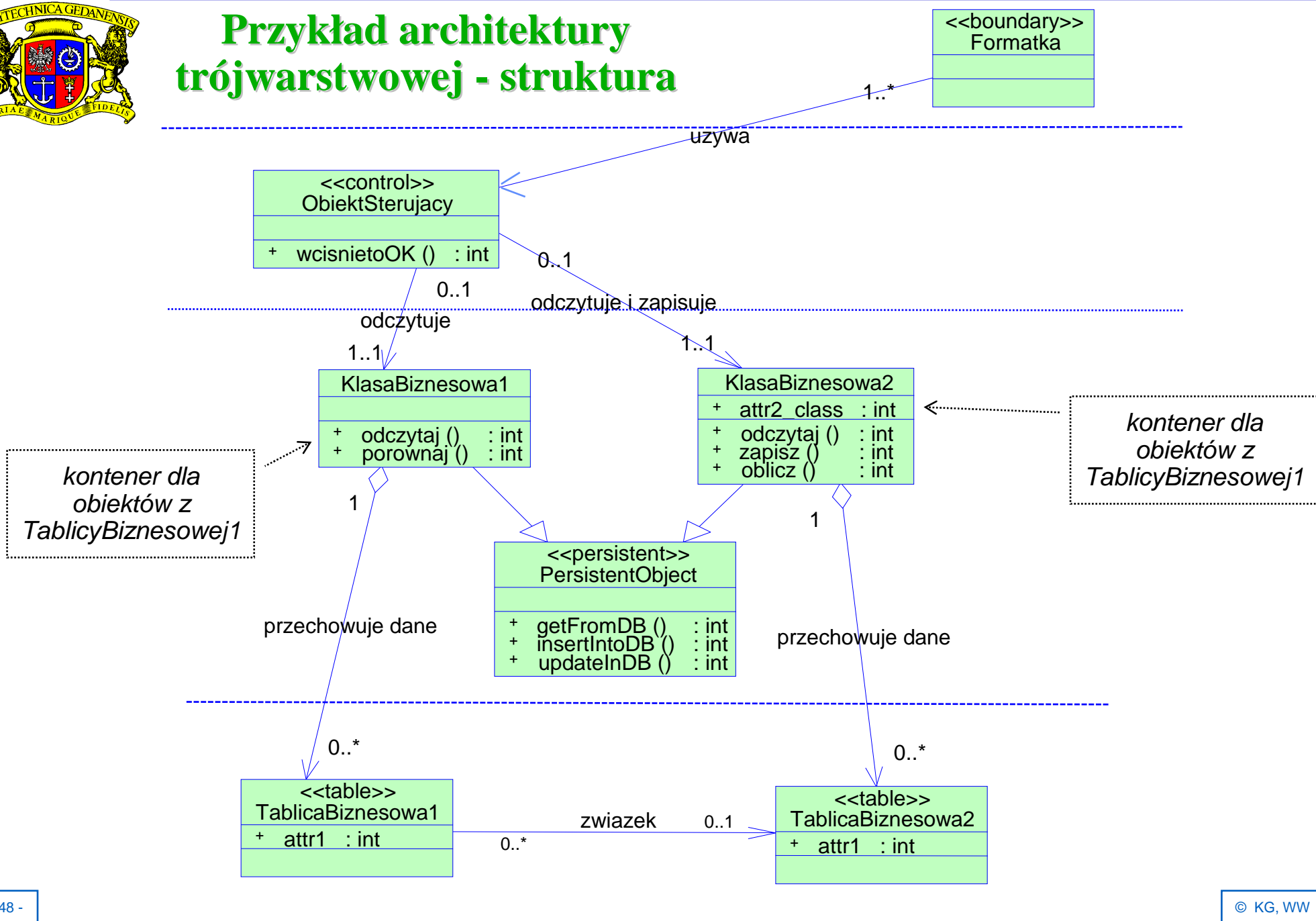
związek

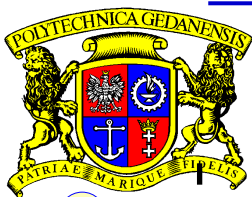
0..1



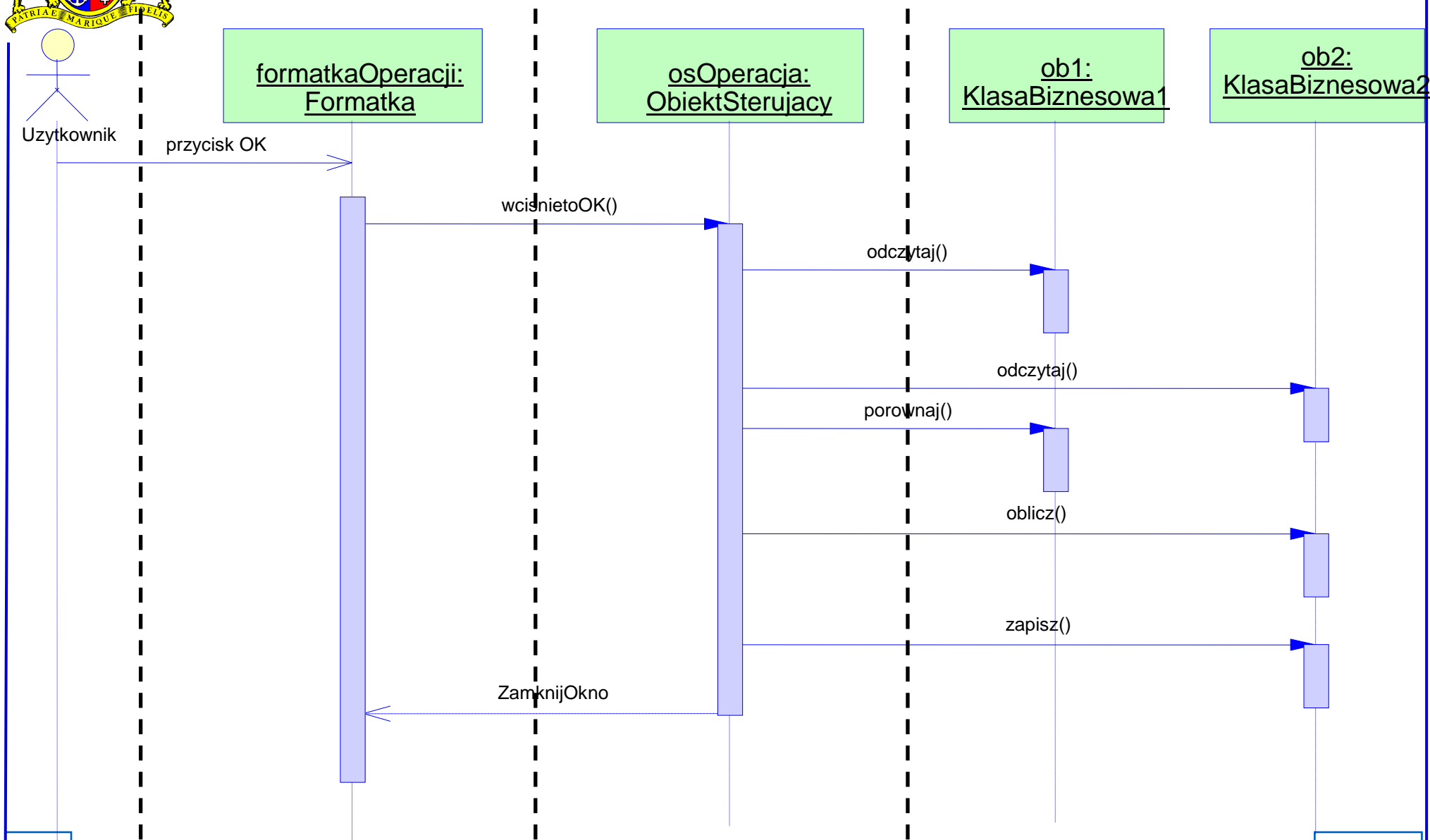


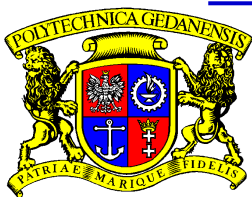
Przykład architektury trójwarstwowej - struktura





Przykład architektury trójwarstwowej - dynamika

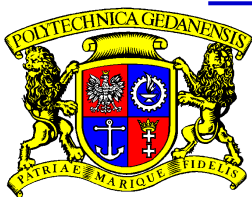




Projektowanie relacyjnej bazy danych

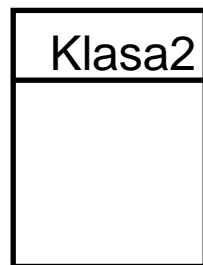
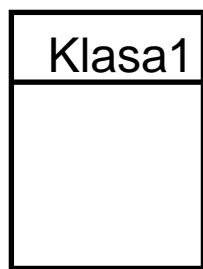
Problemy „ideologiczne”:

Obiekt	Encja
<ul style="list-style-type: none">• Ma tożsamość, niezależną od wartości• Ma zachowanie, określone przez klasę, do której należy• Jest hermetyczny (dostęp poprzez metody)• Wartość ma dowolnie złożoną strukturę	<ul style="list-style-type: none">• Brak tożsamości; identyfikowana kluczem (wartościami atrybutów)• Brak zachowania; tylko dane• Brak hermetyczności (jest przezroczysta)• Określona z góry liczba atrybutów prostych



Model obiektowy → Model relacyjny

1. Klasa → Relacja



Relacja1

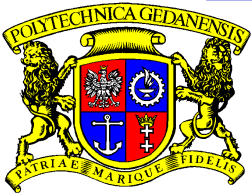
Relacja2

2. Klucz główny relacji

Może trzeba dodać nowy atrybut?

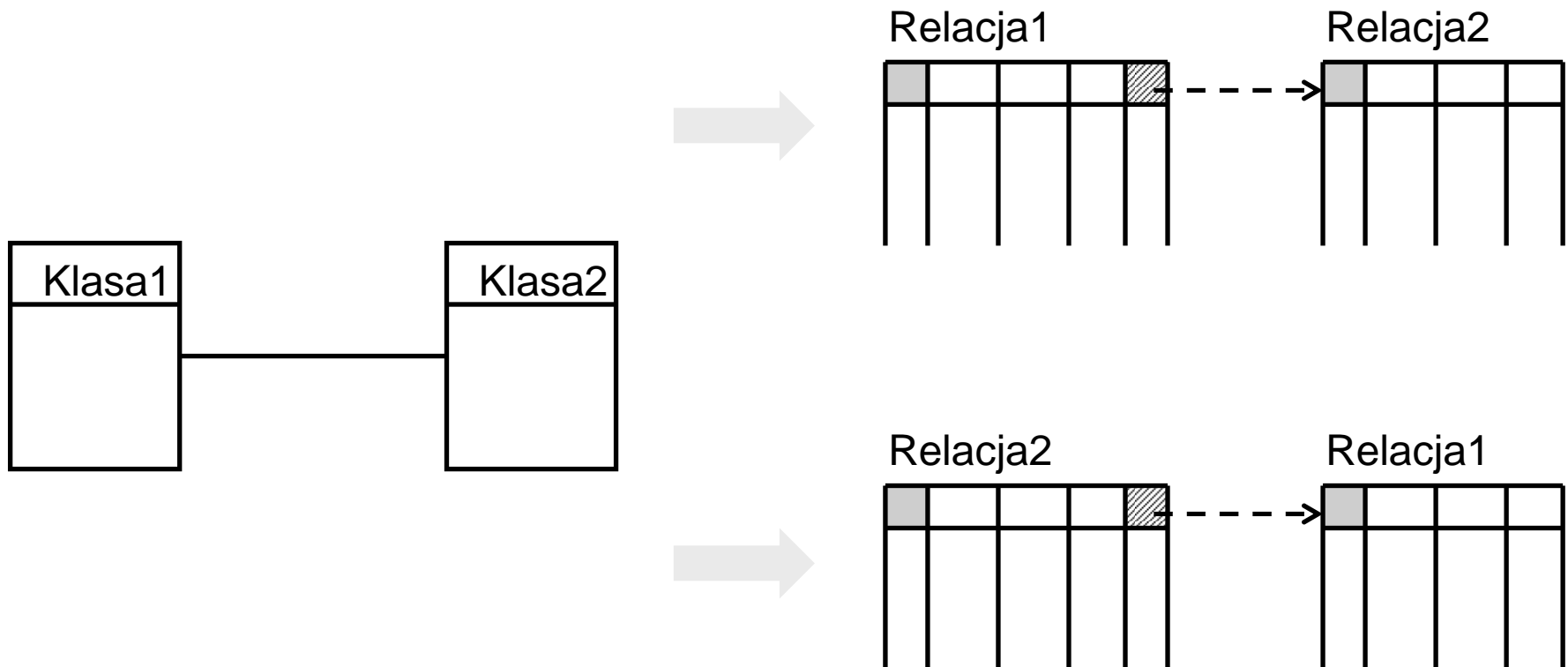
Relacja1

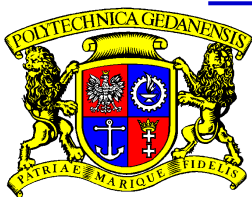
Relacja2



Model obiektowy → Model relacyjny (2)

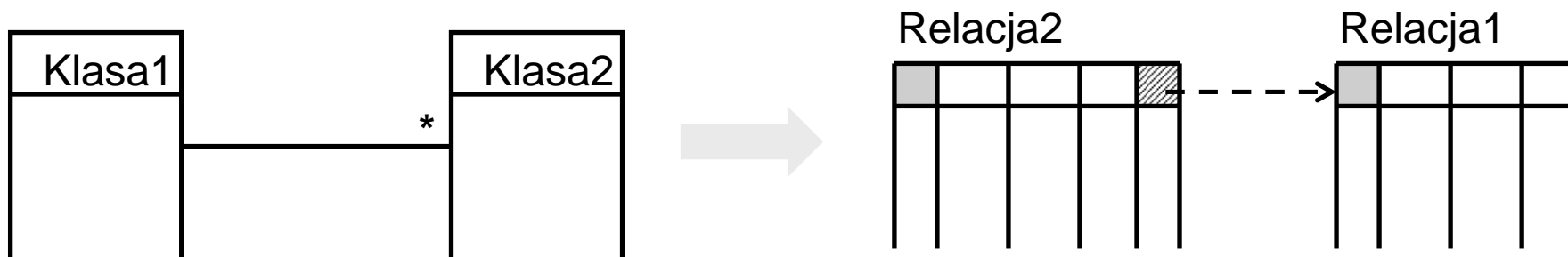
3a. Związki pomiędzy klasami typu 1:1 → Klucze obce

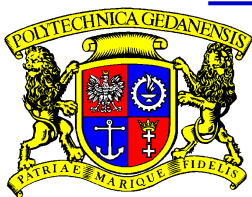




Model obiektowy → Model relacyjny (3)

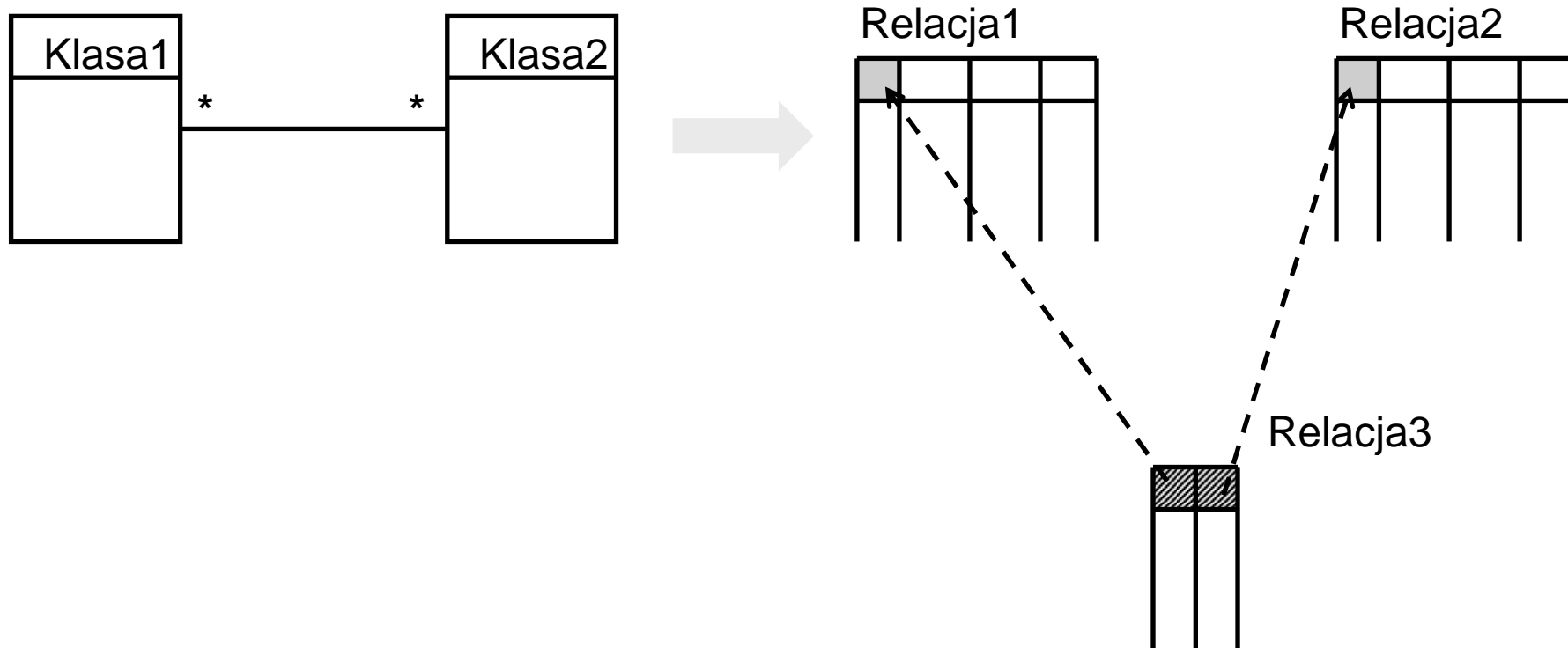
3b. Związki pomiędzy klasami typu 1:n → Klucze obce

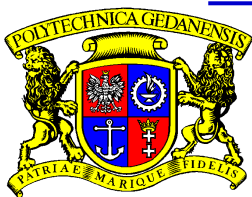




Model obiektowy → Model relacyjny (4)

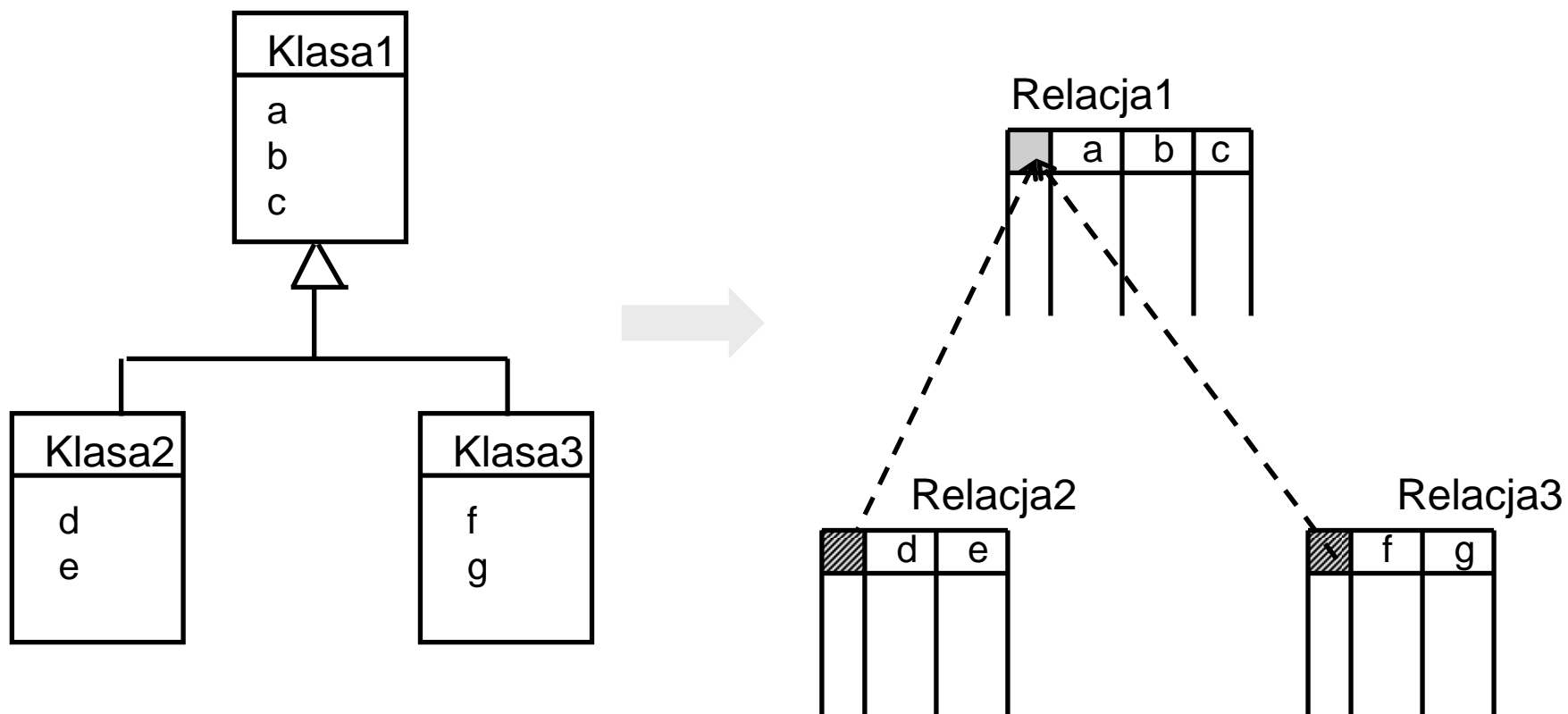
3c. Związki pomiędzy klasami typu $n:m$ → Oddzielna relacja

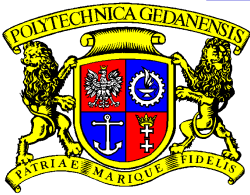




Model obiektowy → Model relacyjny (5)

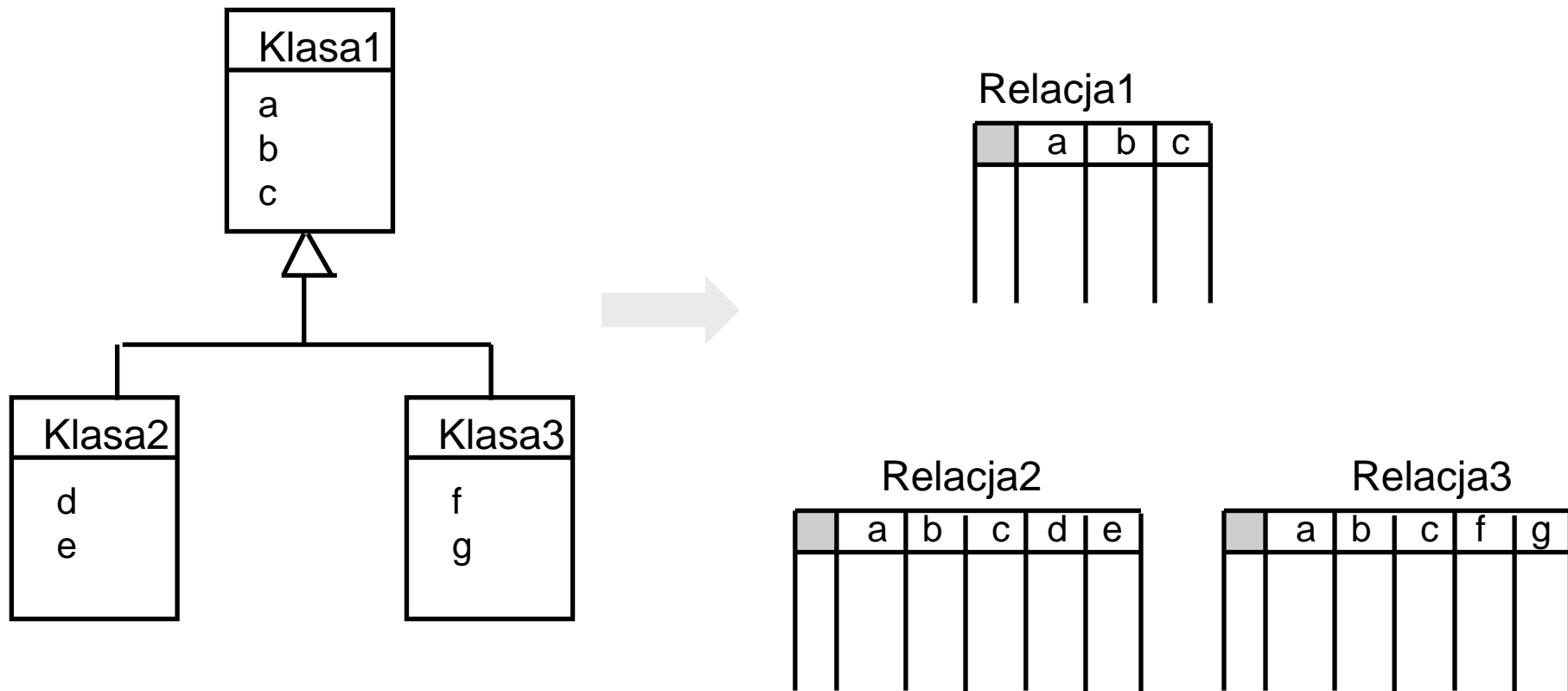
4a. Dziedziczenie → Powiązane relacje

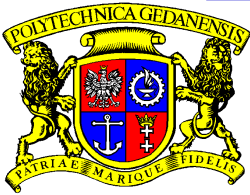




Model obiektowy → Model relacyjny (6)

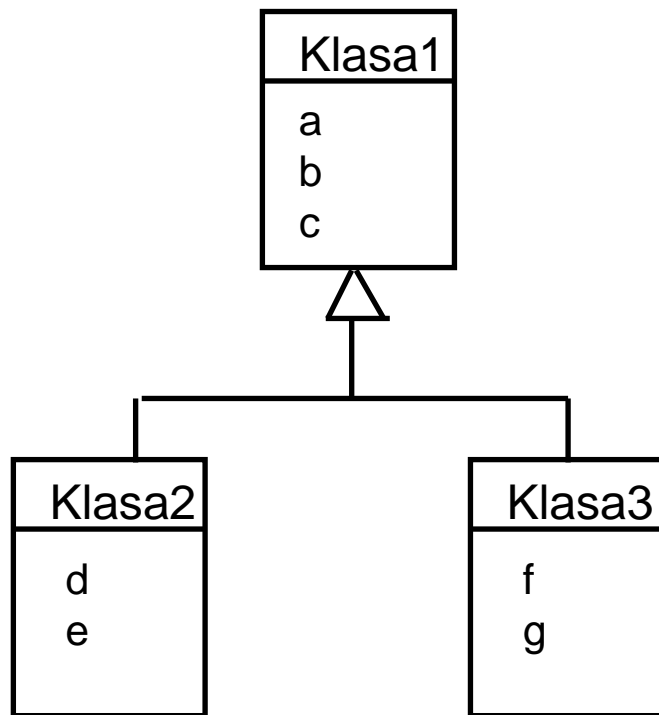
4b. Dziedziczenie → Oddzielne relacje





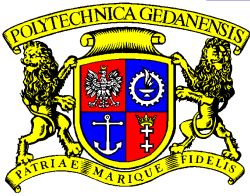
Model obiektowy → Model relacyjny (7)

4c. Dziedziczenie → Jedna relacja z dodatkowym atrybutem „typ”



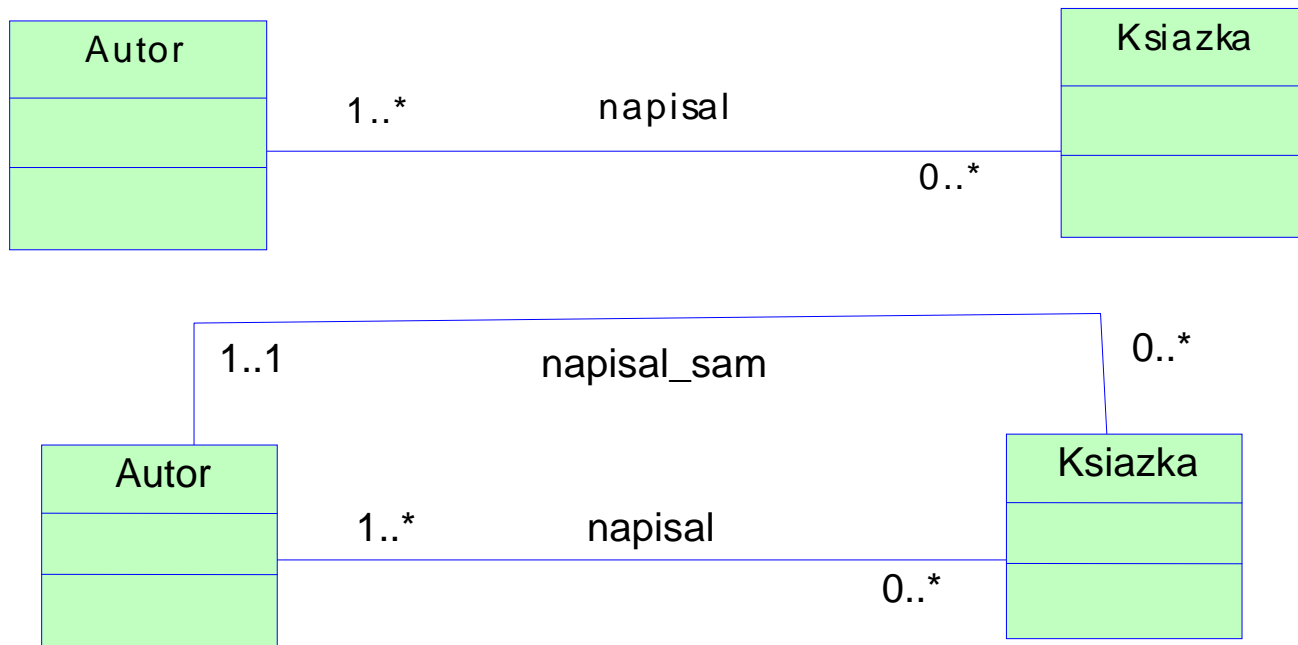
Relacja123

	typ	a	b	c	d	e	f	g

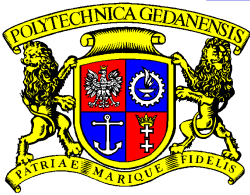


Optymalizacja projektu warstwy danych

- **Optymalizacja projektu związana ze strukturą bazy danych jest wykonywana na poziomie modelu klas**
 - **optymalizacja dostępu do danych**
 - **optymalizacja struktur danych**



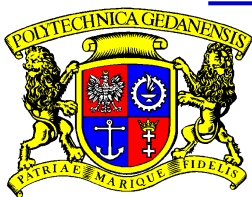
Jaki jest wpływ tej optymalizacji na schemat relacyjnej bazy danych?



Przykład: Baza danych Armatorzy

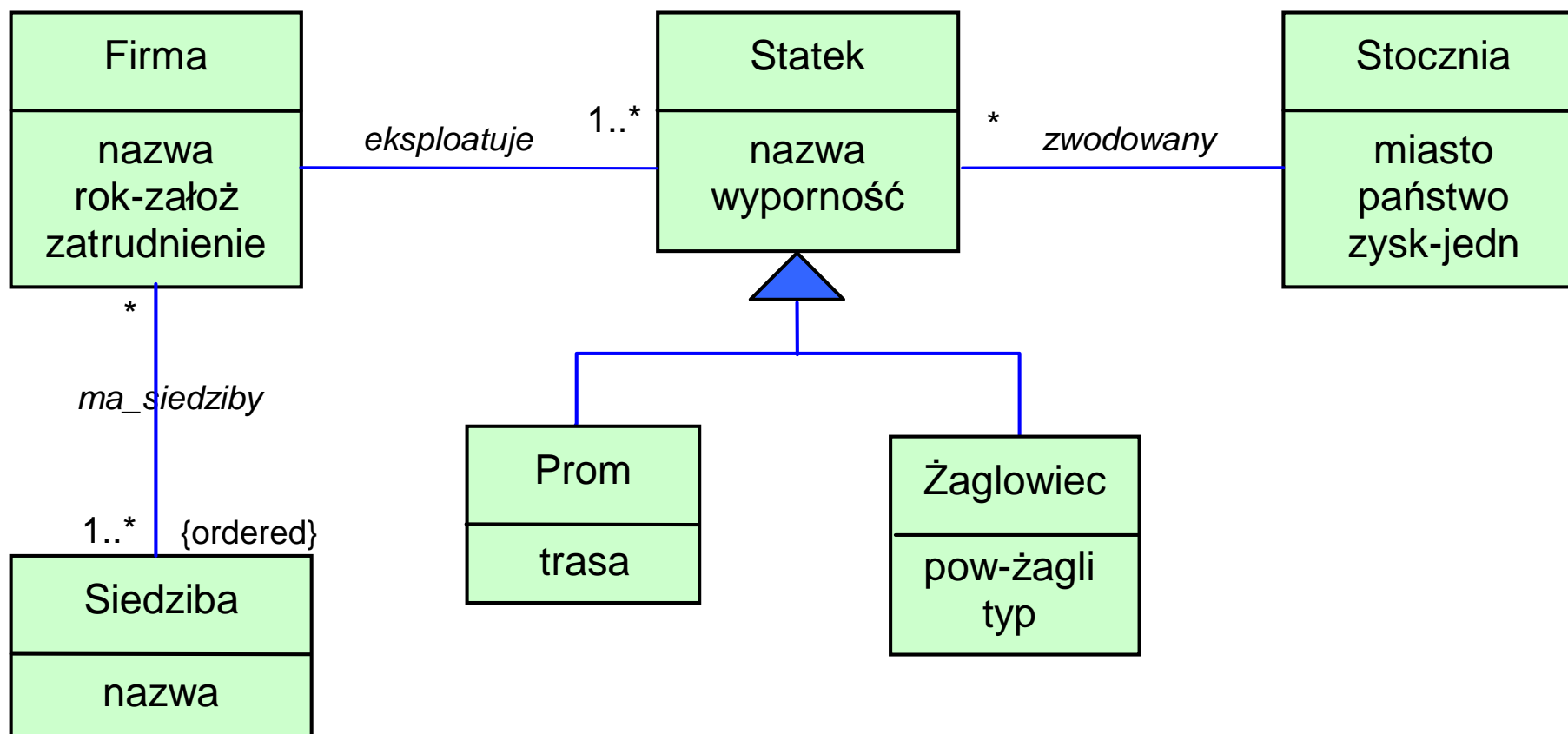
Założenia

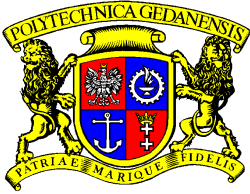
1. System komputerowy ma zarządzać danymi o firmach eksploatujących statki.
2. Każda firma może mieć kilka siedzib (filii).
3. Filie jednej firmy są uporządkowane według pewnego kryterium ważności.
4. Każda siedziba ma swoją unikatową nazwę.
5. W jednej siedzibie może mieścić się wiele firm.
6. Dla każdej firmy chcemy znać: nazwę, rok założenia, liczbę zatrudnionych, a także wiedzieć, gdzie ma swoje siedziby i jakie statki eksploatuje.
7. Każdy statek eksploatowany przez firmę ma nazwę i wyporność.
8. Wśród statków wyróżniamy promy i żaglowce.
9. Każdy prom ma określoną trasę kursowania.
10. Każdy żaglowiec ma typ i powierzchnię żagli.
11. Każdy statek jest eksploatowany przez jednego armatora (firmę).
12. Każdy statek został zwodowany w określonej stoczni.
13. Dla każdej stoczni chcemy znać: jej lokalizację, określoną przez miasto i państwo, parametr ekonomiczny (średni zysk z wyprodukowania jednego statku, w procentach) oraz to, jakie statki (spośród tych, które są eksploatowane przez firmy) w niej zwodowano.



Przykład: Baza danych Armatorzy

Model klas





Przykład: Baza danych Armatorzy

Schemat RBD

Firmy (ident, nazwa, rok-założ, zatrudnienie)

Statki (ident, nazwa, producent REF Stocznie, wyporność, firma REF Firmy)

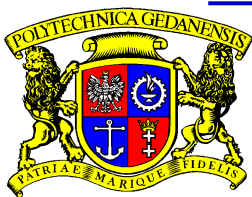
Promy (ident REF Statki, trasa)

Żaglowce (ident REF Statki, pow-żagli, typ)

Stocznie (ident, miasto, państwo, zysk-jedn)

MaSiedz(firma REF Firmy, nazwa, numer)

Klasa Siedziba została osadzona w relacji MaSiedz, gdyż ma tylko jeden atrybut.



Przykład: Baza danych Armatorzy

Schemat RBD (SQL 92)

```
create table Firmy
(ident      char(3) PRIMARY KEY,
 nazwa     varchar(128),
 rok_zaloz integer,
 zatrudnienie integer);

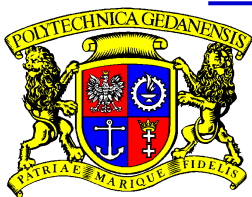
create table Stocznie
(ident      char(3) PRIMARY KEY,
 miasto    varchar(32),
 panstwo   varchar(32),
 zysk_jedn real);

create table Statki
(ident      char(3) PRIMARY KEY,
 nazwa     varchar(32),
 producent char(3)
           REFERENCES Stocznie,
 wypornosc integer,
 firma     char(3)
           REFERENCES Firmy);
```

```
create table Zaglowce
(ident      char(3) PRIMARY KEY
           REFERENCES Statki,
 pow_zagli  real,
 typ        char(16));

create table Promy
(ident      char(3) PRIMARY KEY
           REFERENCES Statki,
 trasa      varchar(128));

create table MaSiedz
(firma      char(3)
           REFERENCES Firmy,
 nazwa     varchar(128),
 numer     smallint,
 PRIMARY KEY (firma,numer));
```



Przykład: Baza danych Armatorzy

Przykładowe wystąpienie RBD

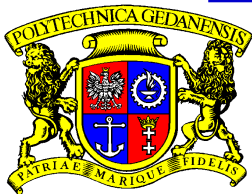
Firmy	ident	nazwa	rok-założ	zatrudnienie
	001	RYBITWA	1918	289
	002	NEPTUN	1969	119
	003	WODNIK	1969	257
	004	PEGAZ	1918	12

MaSiedz	firma	nazwa	numer
	001	GODYNIA	1
	001	GDAŃSK	2
	002	GDAŃSK	1
	003	GDAŃSK	1
	004	GODYNIA	1
	004	SZCZECIN	2
	004	GDAŃSK	3

Statki	ident	nazwa	producent	wyporność	firma
	001	ORZEŁ	001	20 000	001
	002	SOKÓŁ	002	18 000	002
	003	WILK	002	25 000	004
	004	ŁABĘDŹ	002	20 000	001
	005	MEWA	003	15 000	003
	006	KONDOR	003	18 000	004

Promy	ident	trasa	Żaglowce	ident	pow-żagli	typ
	004	Px – Py		006	1 200	SZKUNER
	005	Py – Pz				

Stocznie	ident	miasto	państwo	zysk-jedn
	001	PLYMOUTH	ANGLIA	11,13
	002	SZCZECIN	POLSKA	12,10
	003	HAMBURG	NIEMCY	11,59

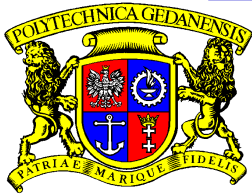


Projekt algorytmów

- **Analiza: punkt widzenia usługi (CO?)**
Projekt: punkt widzenia realizacji (JAK?)
- **Projekt algorytmów to proces iteracyjny kończący się, gdy wszystkie operacje są łatwe („oczywiste”) do zaimplementowania**
- **Projekt algorytmów obejmuje**
 - wybór algorytmów
 - wybór struktur danych dla algorytmów
 - definicje klas pomocniczych (wewnętrznych w danym pakiecie)
 - przypisanie operacji do klas

**Każda operacja wyspecyfikowana
w modelu klas powinna być przedstawiona
w postaci algorytmu (pseudokodu)**



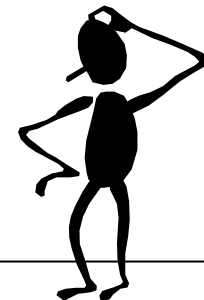


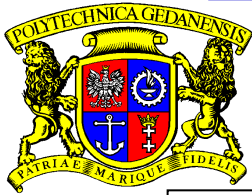
Wybór algorytmów

- **Kryteria wyboru**

- złożoność obliczeniowa (CZAS)
- złożoność pamięciowa (PAMIĘĆ)
- zbiór dostępnych danych (LOGIKA)
- łatwość implementacji i przejrzystość (NAKŁAD PRACY)
- modyfikowalność (A CO POTE M?)

A co, jeśli algorytm nie jest znany?

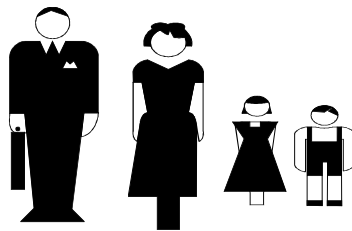




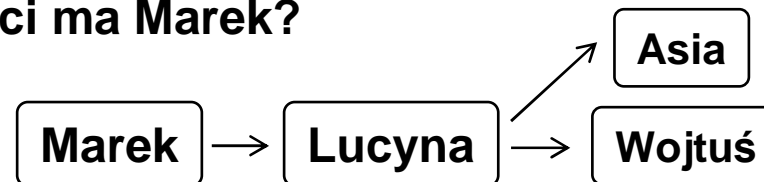
Wybór algorytmów

- Algorytmy „proste”

- “CO” jednocześnie definiuje “JAK” (nalicz_vat())
- operacje dostępu do atrybutów (put_, get_)
- dostęp do obiektów przez nawigację w sieci powiązań między obiektami



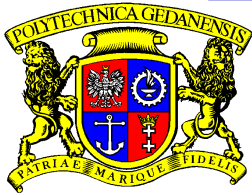
Ile dzieci ma Marek?



- Algorytmy „trudne”

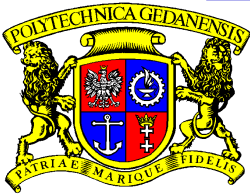
- procesy bez proceduralnej specyfikacji

\sqrt{N} to taka liczba, że $\sqrt{N} * \sqrt{N} = N$



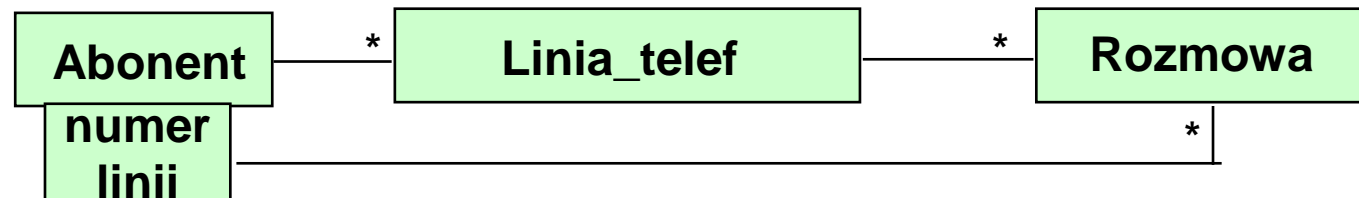
Wybór struktur danych

- **Wybór algorytmu narzuca wymagania na struktury danych**
- **Dodane struktury nie dodają nowej informacji do modelu analitycznego, lecz organizują go w sposób wygodny dla działania algorytmu**
- **Klasy kolekcyjnych typów danych**
 - listy, zbiory, drzewa ...
 - typy związane z implementacją dostępu do danych (referencja, kolekcja)
- **Użycie bibliotek**
- ...

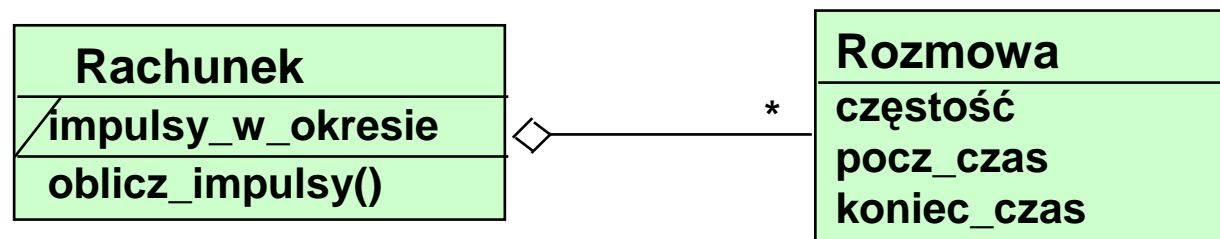


Optimalizacja projektu

- Dodanie związków nadmiarowych (wyluczanych)
 - czy istnieje szczególne uporządkowanie sieci powiązań optymalizujące krytyczne części systemu?
 - należy rozważyć charakter („wzorzec”) i częstości dostępu

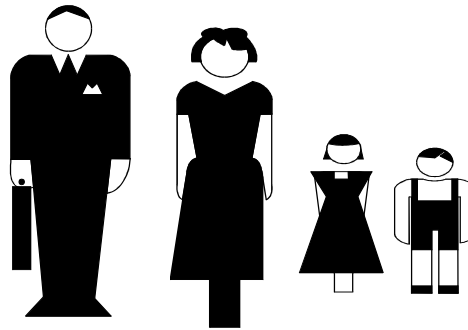


- Indeksy dla zapytań o małym stopniu trafienia
- Przechowywanie atrybutów nadmiarowych (wyluczanych)

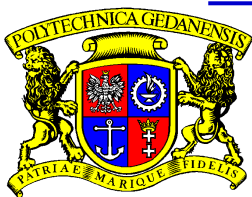


Projekt związków

- **W świecie rzeczywistym związki między obiektami są dwukierunkowe**

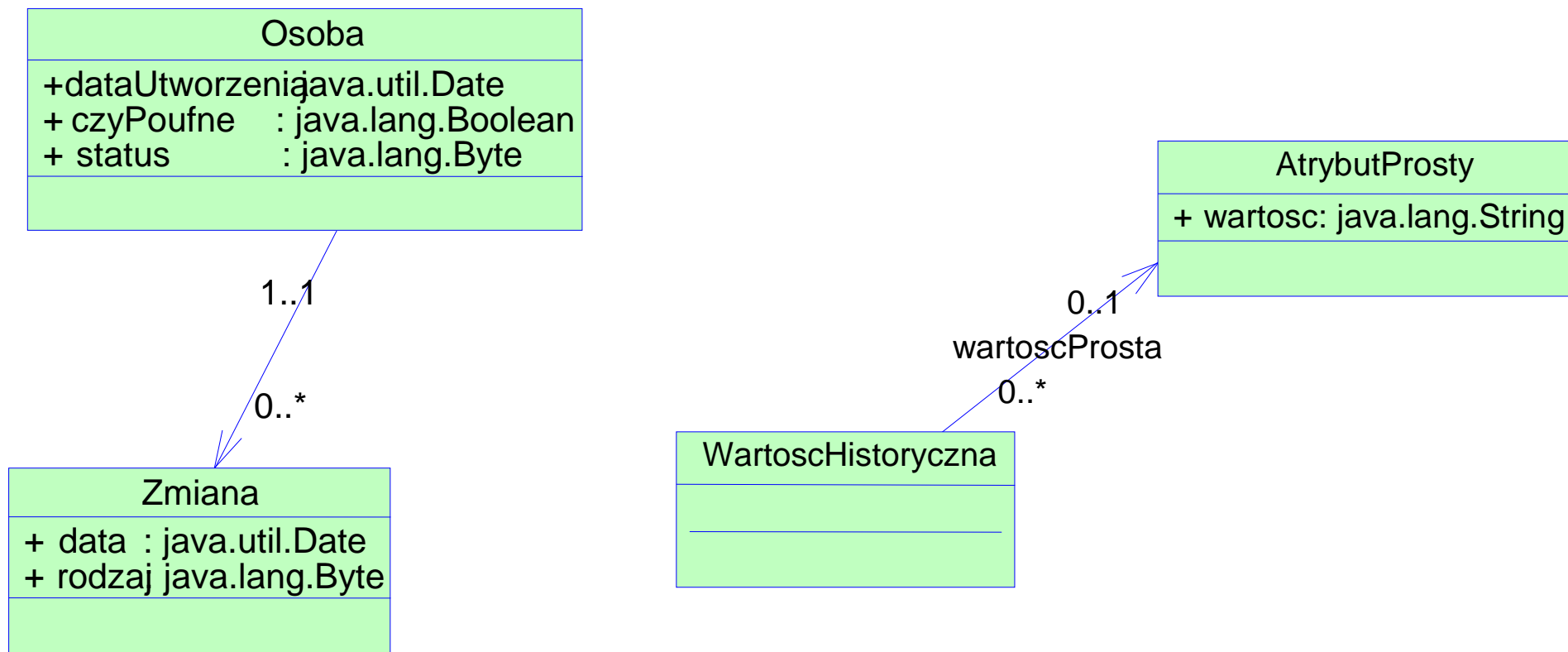


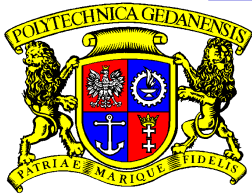
- **W celu identyfikacji związków jednokierunkowych w systemie należy przeanalizować kierunki przejść powiązań w typowych operacjach**
 - prostsza implementacja
 - słabsza modyfikowalność



Projekt związków

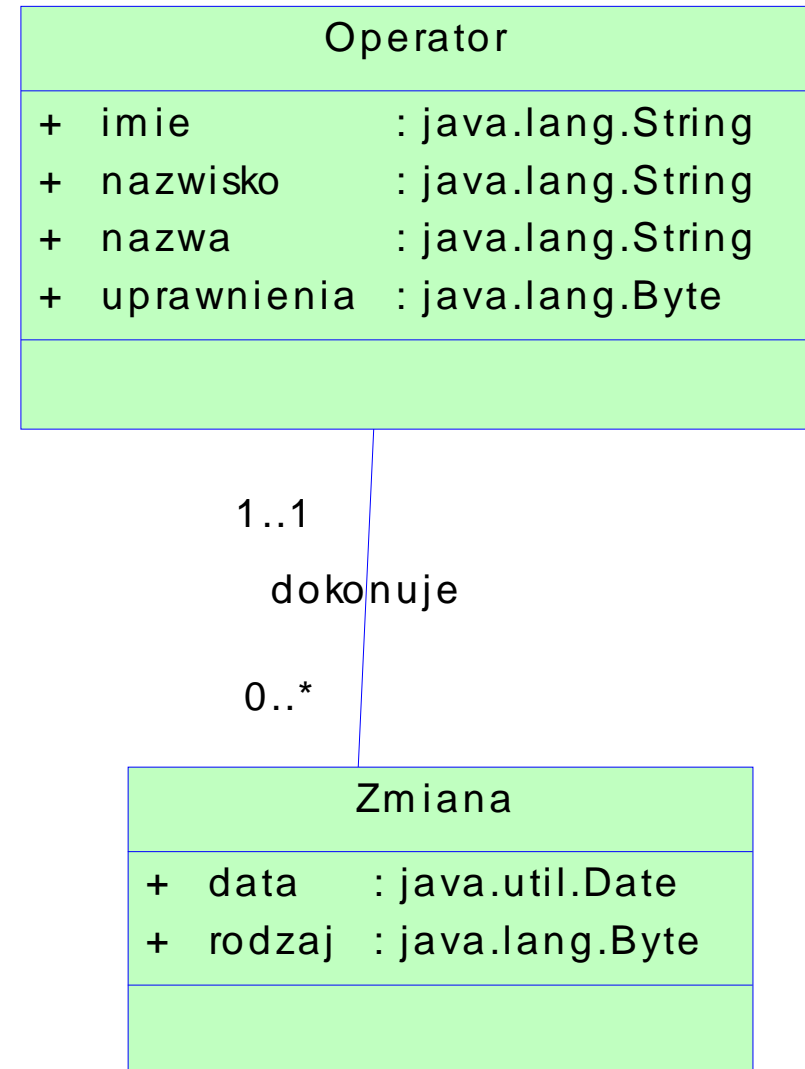
- Związek jednokierunkowy, w zależności od jego liczności i kierunku zapytań, jest realizowany jako wskaźnik lub zbiór (lista) wskaźników.

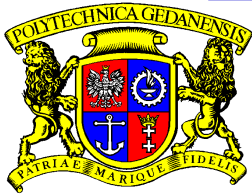




Projekt związków

- **Realizacja związku dwukierunkowego**
 - jak jednokierunkowe i przeszukiwanie (kosztowne!)
 - podwójne jednokierunkowe (trudna aktualizacja!)

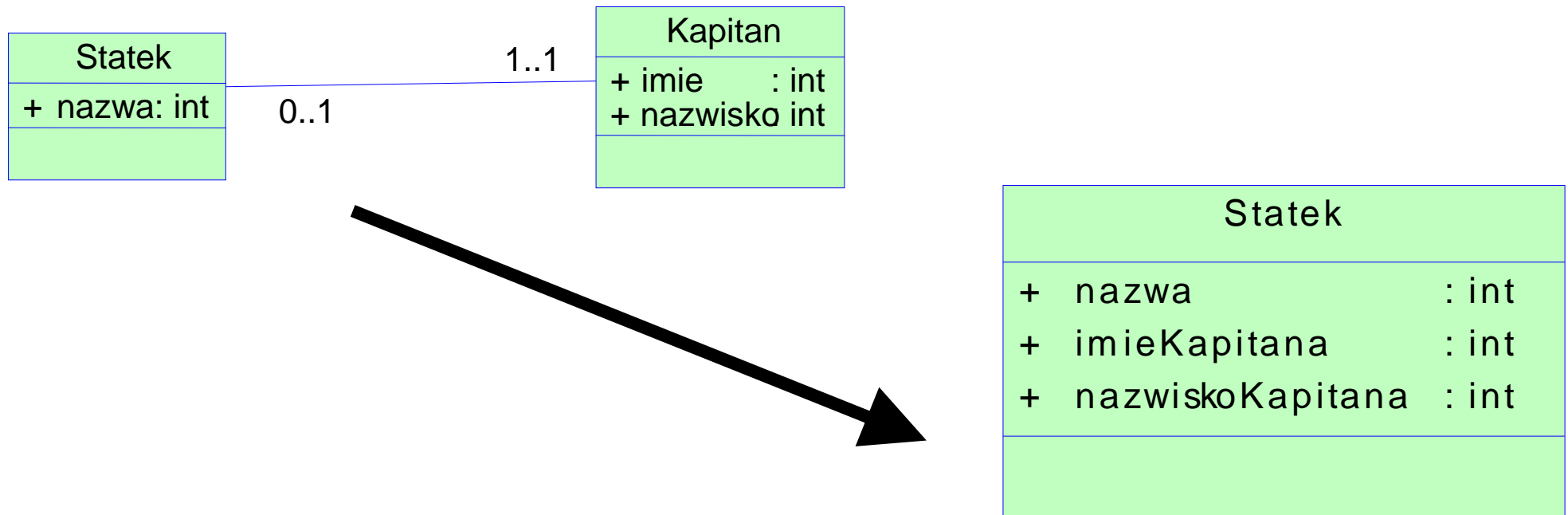


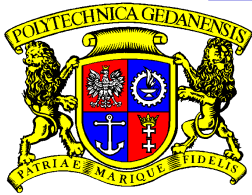


Projekt związków

- **Związki 1:1**

- istnienie związku 1:1 zawsze powinno zwrócić uwagę projektanta
 - co oznacza związek 1:1?
 - jednemu obiektowi klasy A odpowiada jeden i tylko jeden obiekt klasy B
 - jednemu obiektowi klasy B odpowiada jeden i tylko jeden obiekt klasy A
- w wielu przypadkach związki 1:1 zamienia się na klasy osadzone

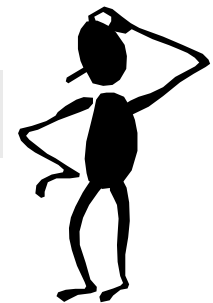


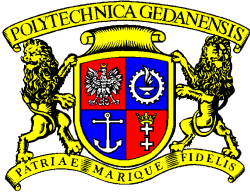


Projektowanie klas - produkty

- **Szczegółowe diagramy klas**
- **Szczegółowy projekt bazy danych**
- **Szczegółowe diagramy sekwencji dla wszystkich przypadków użycia i tych metod, w których występuje skomplikowana interakcja z innymi obiektami**
- **Określenie realizacji interfejsów występujących na diagramach klas**

Czy programista ma wystarczające informacje do pracy?





Diagramy UML na etapach wytwarzania systemu

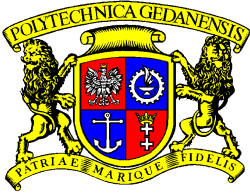
- **Specyfikowanie wymagań**
 - opisy słowne, procesy biznesowe, przypadki użycia

Diagramy:

- diagramy hierarchii i przebiegu procesów biznesowych
- diagramy przypadków użycia
- wstępne diagramy klas
- wstępne diagramy czynności



Znajdź wspólny język z klientem!



Diagramy UML na etapach wytwarzania systemu (2)

- **Modelowanie**

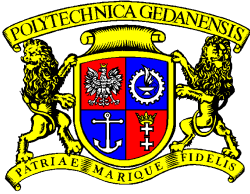
- tworzone są modele z dziedziny problemowej, obejmujące:
klasy, obiekty, interakcje;
- uszczegóławiane są (lub tworzone, jeśli jeszcze nie istnieją) przypadki użycia
- definiowane są klasy kandydujące do umieszczenia w modelu, po czym eliminowane są klasy zbędne i dodawane pominięte („burza mózgów”)
- modelowane są statyczne zależności pomiędzy klasami (jako związki różnych typów)
- modelowane jest zachowanie się obiektów: analizowane są scenariusze realizacji poszczególnych przypadków użycia na diagramach interakcji

Diagramy:

- diagramy klas i obiektów
- diagramy sekwencji i współpracy
- diagramy czynności
- diagramy stanów



Położ nacisk na dziedzinę problemową - nie na rozwiązania techniczne!



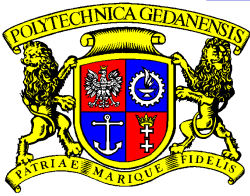
Diagramy UML na etapach wytwarzania systemu (3)

• Projektowanie

- uszczegóławiany jest model systemu (projekt klas)
- określana jest architektura systemu:
 - klasy „pakowane” są w pakiety (moduły) funkcjonalne
- identyfikowana jest współbieżność w systemie, identyfikowane są komunikaty asynchroniczne oraz określane są techniki obsługi współbieżności i współużytkowania zasobów
- definiowane są formaty danych wejściowych i wyjściowych
- określane są gotowe komponenty (biblioteki, systemy oprogramowania) potrzebne do realizacji systemu
- klasy trwałe przekształcane są w schemat bazy danych
- definiowana jest obsługa sytuacji awaryjnych i wyjątkowych
- komponenty oprogramowania systemu przydzielane są do komponentów sprzętowych

Diagramy:

- diagramy klas
- diagramy sekwencji i kooperacji
- diagramy czynności
- diagramy stanów
- diagramy komponentów
- diagramy rozmieszczenia



Diagramy UML na etapach wytwarzania systemu (4)

- **Implementacja**

Diagramy z fazy analizy i projektowania są uszczegóławiane i aktualizowane (np. diagramy komponentów). Zwykle nie są tworzone nowe diagramy.

- **Testowanie**

Diagramy z fazy analizy i projektowania służą jako odniesienie dla przypadków testowych:

- diagramy rozmieszczenia, sekwencji i współpracy służą do testowania integracyjnego
- diagramy przypadków użycia służą do testowania walidacyjnego i akceptacyjnego