

**Koncepcja, projekt i
implementacja
wielowarstwowego systemu
informatycznego
Wykład 6**

Zofia Kruczkiewicz

Literatura

1. Roger S. Pressman, Praktyczne podejście do oprogramowania, WNT, 2004
2. Stephen H. Kan, Metryki i modele w inżynierii jakości oprogramowania, Mikom, PWN, 2006
3. Jacobson, Booch, Rumbaung, The Unified Software Development Process, Addison Wesley, 1999
4. Shalloway A., Trott James R., Projektowanie zorientowane obiektowo. Wzorce projektowe. Gliwice, Helion, 2005
5. Robert C. Martin, Micah Martin, Agile Programowanie zwinne. Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#, Helion 2019
6. Robert C. Martin Czysta architektura. Struktura i design oprogramowania. Przewodnik dla profesjonalistów. Helion, 2022
7. D. Alur, J. Crupi, D. Malks, Core J2EE. Wzorce projektowe
8. [Java Platform, Enterprise Edition The Java EE Tutorial Java Platform, Enterprise Edition, Release 8](#)

Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. **5 zasad programowania solidnego (solid) [5]**
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
6. Przykłady architektury wielowarstwowej aplikacji typu **EE** (p.2) . Wykonanie aplikacji typu **EE**. *Warstwa biznesowa*: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów (EIS)* - baza danych w systemie baz danych **Apache Derby**
8. Utworzenie obiektowego modelu danych do utrwalania **ORM**
9. *Warstwa integracji*. Zastosowanie wzorca projektowego typu **Domain Store** w technologii **JPA (Java Persistence)** na platformie **Java EE**
10. *Warstwa prezentacji* - **JSF**
11. Dodatek

Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego

Definicja systemu informatycznego (wykład 1)

(na podstawie Paul Beynon_Davies, Inżynieria systemów informacyjnych)

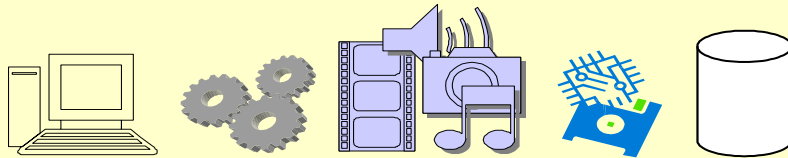
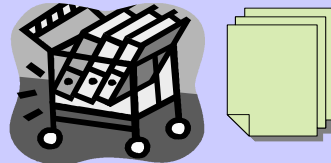
Nieformalny system informacyjny

zasoby osobowe – ludzie,
zasoby sprzętowe



Formalny system informacyjny:

procedury zarządzania,
bazy wiedzy



Techniczny system informacyjny:

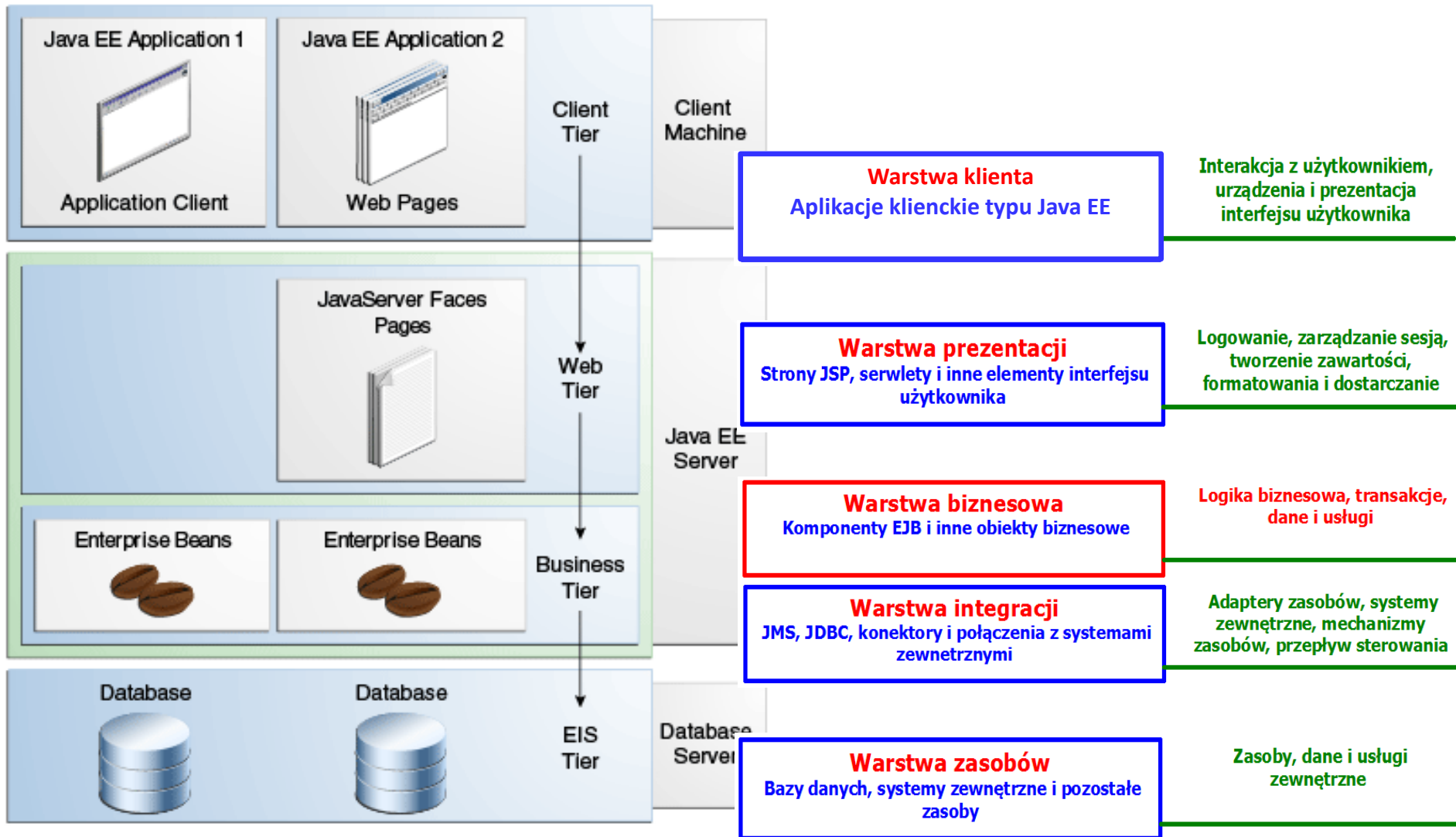
- Sprzęt
- Oprogramowanie
- Bazy danych, bazy wiedzy

System informatyczny jest to zbiór powiązanych ze sobą elementów **nieformalnych, formalnych i technicznych**, którego funkcją jest przetwarzanie danych przy użyciu techniki komputerowej

Techniczny system informacyjny

- zorganizowany zespół środków technicznych (komputerów, **oprogramowania**, urządzeń teletransmisyjnych itp.)
- służący do gromadzenia, przetwarzania i przesyłania informacji.

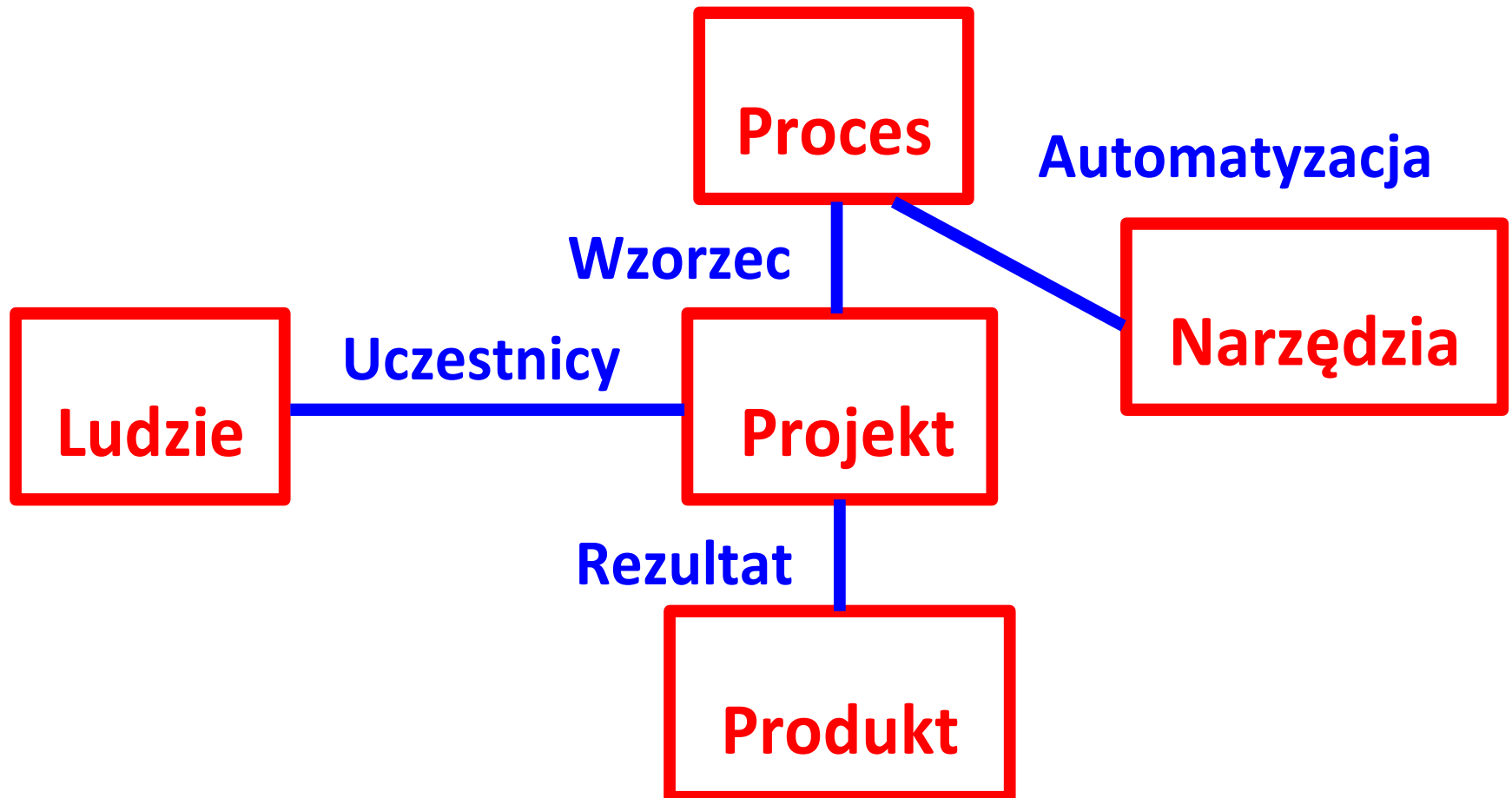
Pięciowarstwowy model logicznego rozdzielania zadań [7] (wykład 1)



Komponent – produkt do budowy warstw

- skompilowany moduł programowy,
- funkcjonalność dostarczana za pomocą interfejsu,
- zdolny do współdziałania z innymi komponentami oraz innymi częściami systemu informatycznego.

Elementy tworzenia oprogramowania – struktura (wykład 1) – dopasowanie procesu wytwarzania do typu produktu



Zagadnienia

- 1. Wielowarstwowa architektura systemu informatycznego**
- 2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego [7] – przykłady architektury**

Wielokryterialna ocena oprogramowania – metryki [2]

Refaktoryzacja to poprawa struktury oprogramowania bez utraty funkcjonalności – w celu poprawy jego **metryk**:

- **wydajności**
- **funkcjonalności**
- **kosztu**
- **jakości oprogramowania:**
 - **Testowalności**
 - **Pielęgnowalności**
 - **Wieloużywalności**
 - **Zrozumiałości**
 - **Stopnia osiągniętej abstrakcji**

Przykłady powiązania metryk kodu z oceną oprogramowania

Metryka	Jakość					Koszt	Funkcjonalność
	Stopień osiągniętej abstrakcji	Wieloużywalność	Zrozumiałość	Pielęgnowalność	Testowalność (niezawodność)		
RFC					+		
CBO					+		
LOC			+	+	+	+	
S/C		+	+	+	+	+	
WMC					+		
DIT			+	+	+		
NOC				+	+		
McCabe			+	+	+	+	
LCOM	+	+	+	+	+		

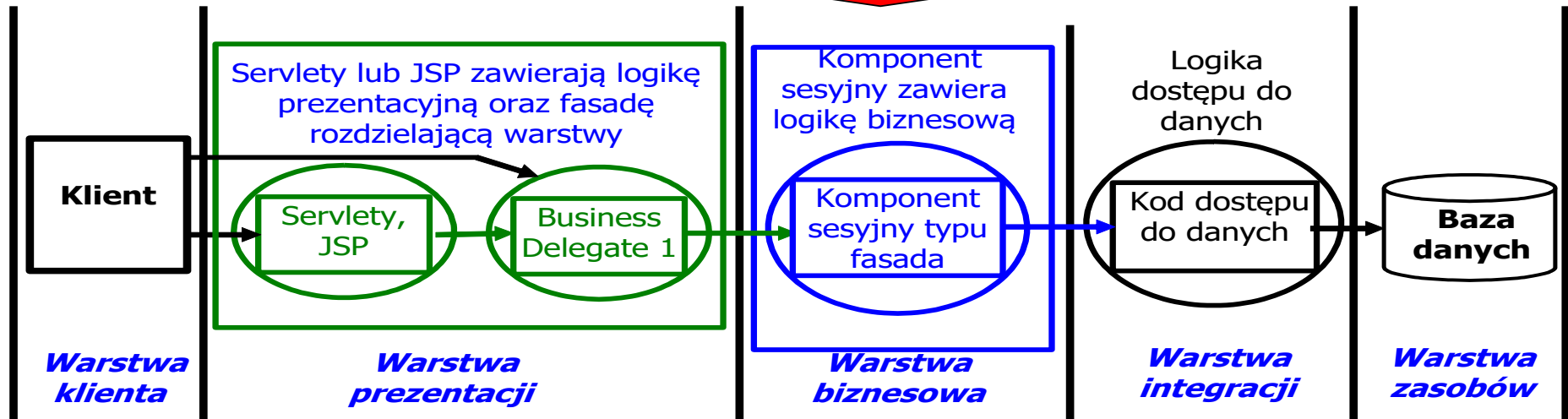
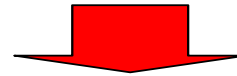
Definicje metryk - [Definicje metryk oprogramowania](#)

Ocena jakości - [Zarządzanie jakością oprogramowania](#)

Narzędzia - [Java Code Quality Tools Recommended by Developers - DZone](#)

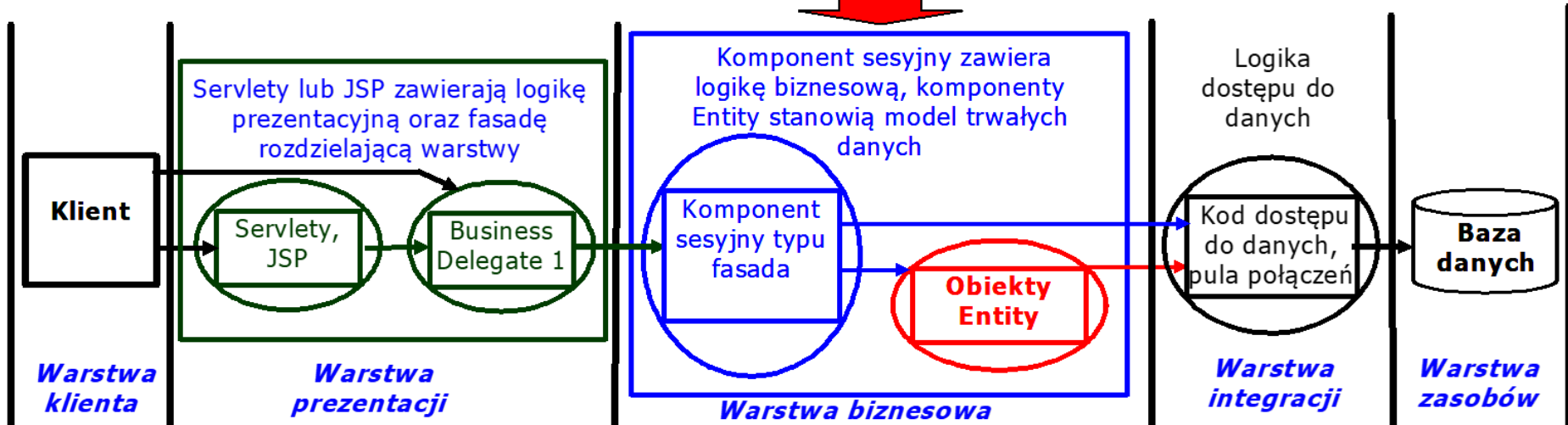
Refaktoryzacja architektury wielowarstwowej - część 1

1. Przeniesienie kodu dostępu do danych logicznie lub fizycznie bliżej rzeczywistego źródła danych ➡ **Warstwa integracji**
2. Przeniesienie kodu logiki przetwarzania z **Warstwy klienta** i **Warstwy prezentacji** ➡ **Warstwy biznesowej** zawierającej **fasadowe komponenty sesyjne** typu **Control**.
Komponenty **Business Delegate** typu **Control** hermetyzują dostęp do **Warstwy biznesowej** z **Warstwy prezentacji**.



Refaktoryzacja architektury wielowarstwowej - część 2

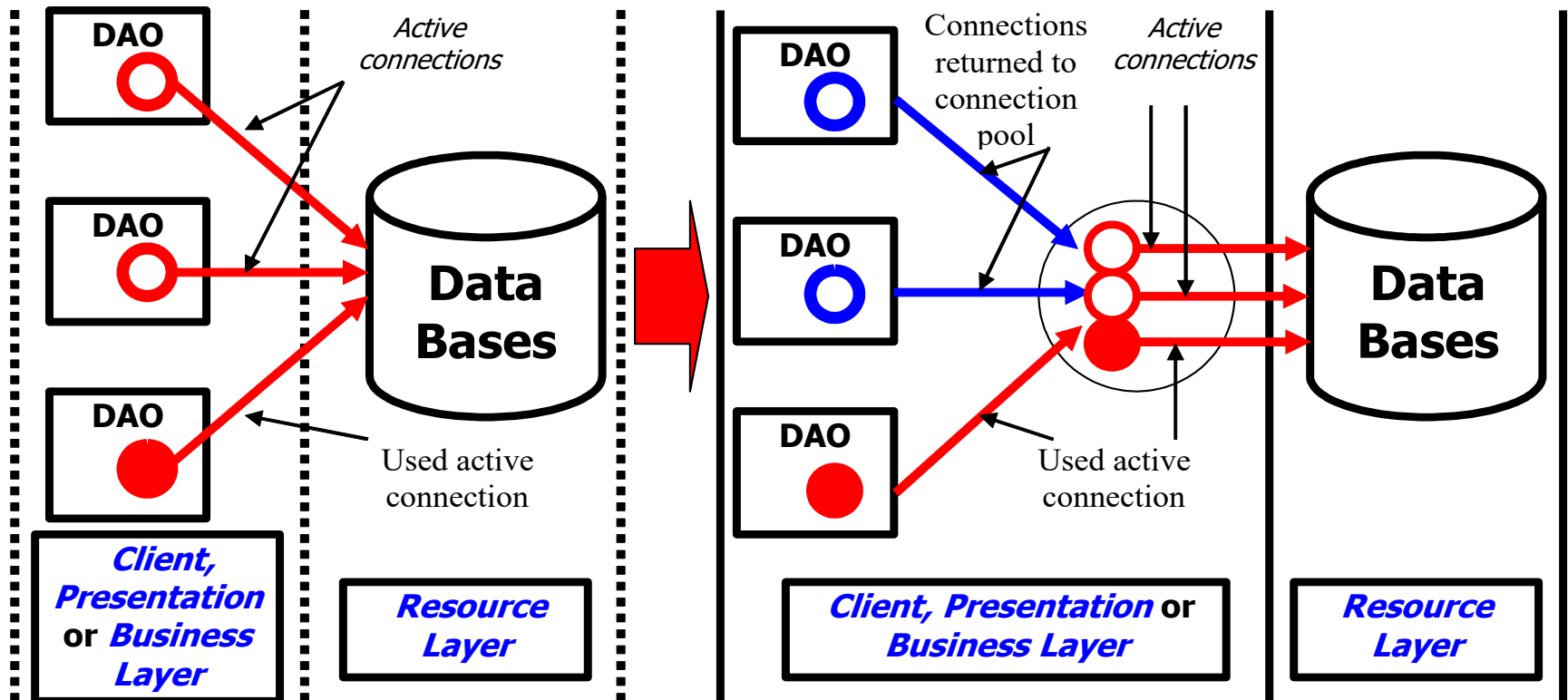
3. Należy zdefiniować obiektowy model danych, który zbudowany jest z **obiektów danych** typu **Entity**



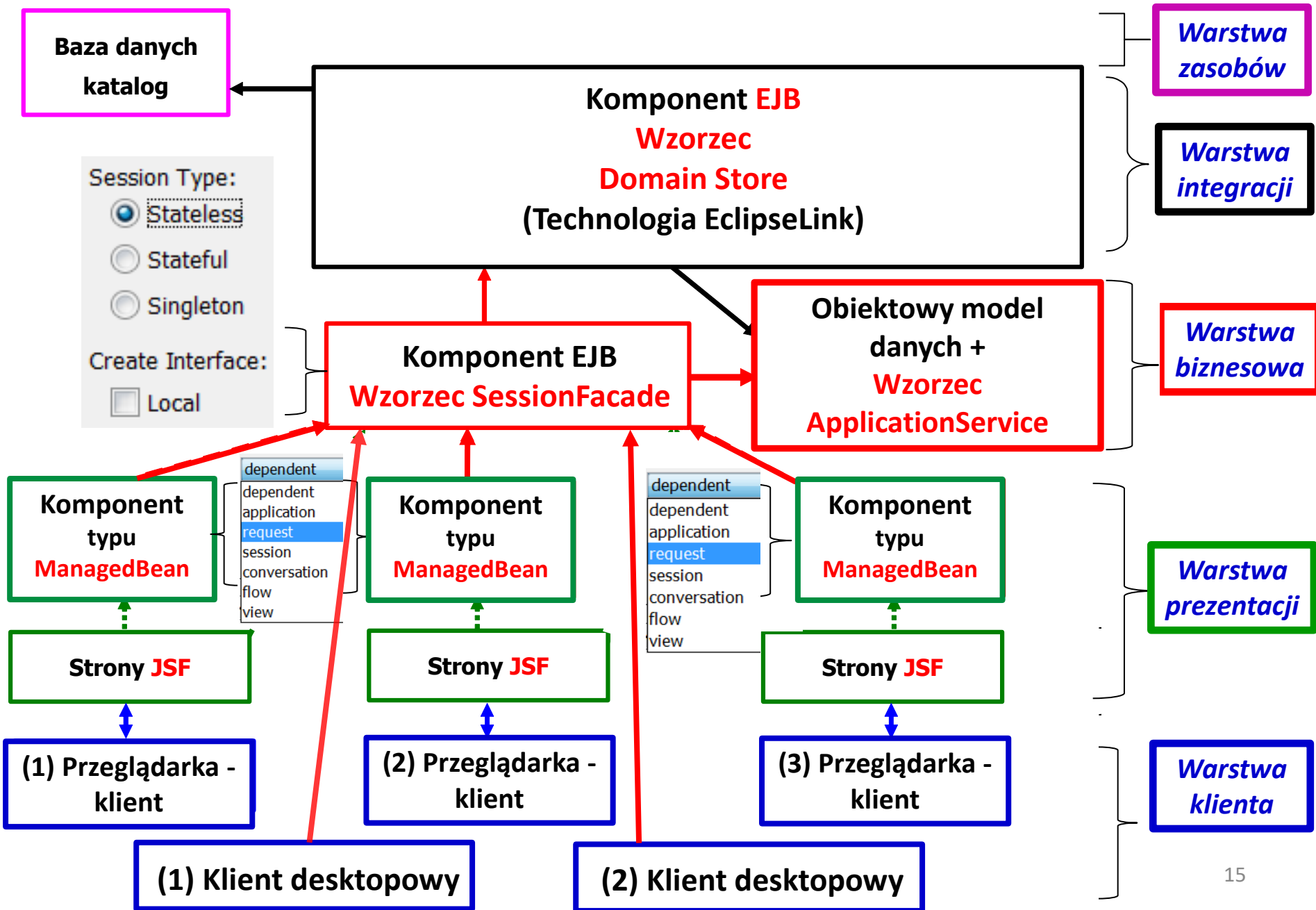
Zarządzanie połączeniami do bazy danych – **pula połączeń**

Połączenia z bazą danych nie są udostępniane, co zmniejsza wydajność i skalowalność.

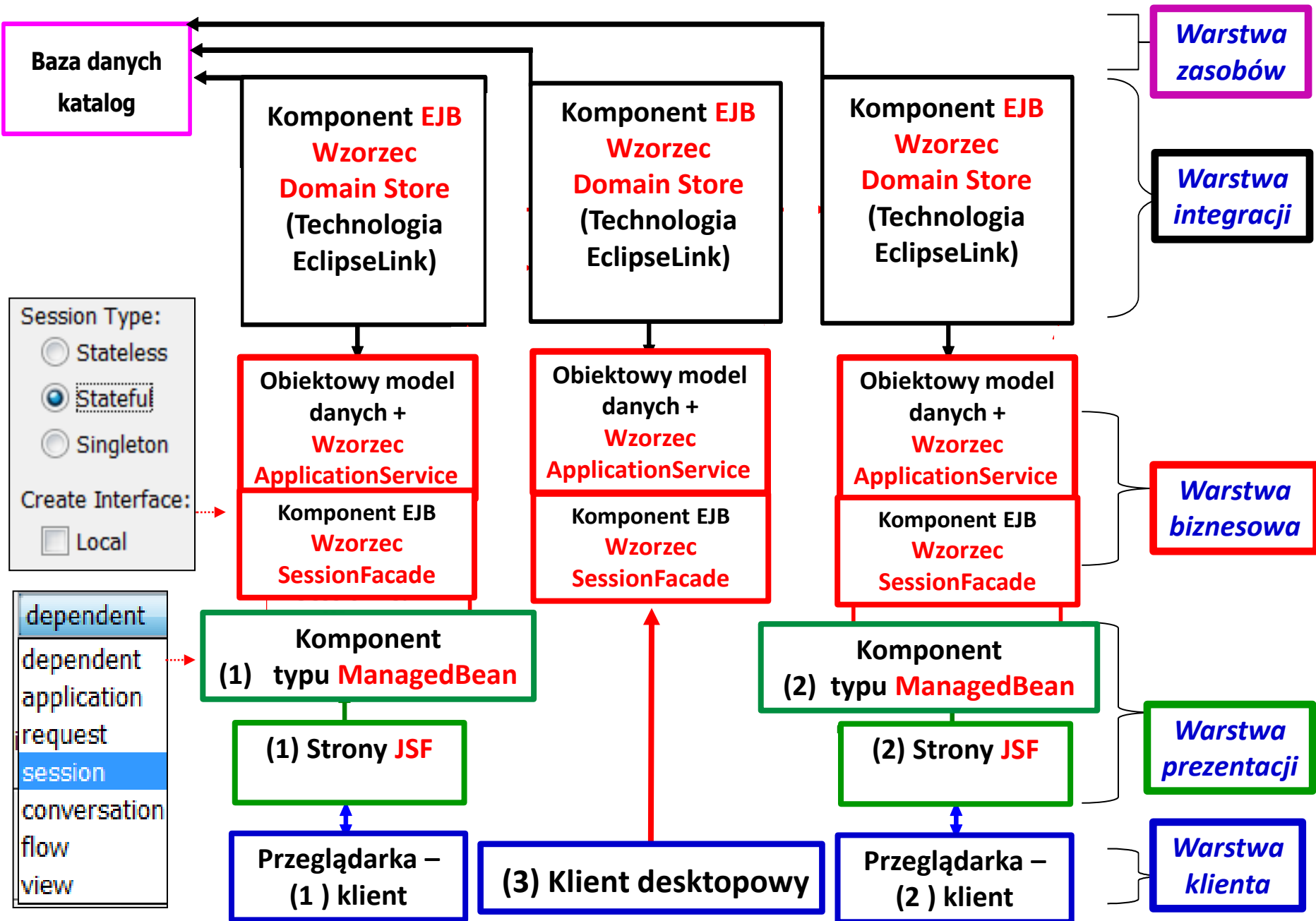
Pula wspólnych połączeń poprawia wydajność i skalowalność aplikacji



Architektura aplikacji pięciowarstwowej - Java EE 8.0 JavaServer Faces



Architektura aplikacji pięciowarstwowej – Java EE 8.0 JavaServer Faces



Zagadnienia

- 1. Wielowarstwowa architektura systemu informatycznego**
- 2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury**
- 3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej [7]**

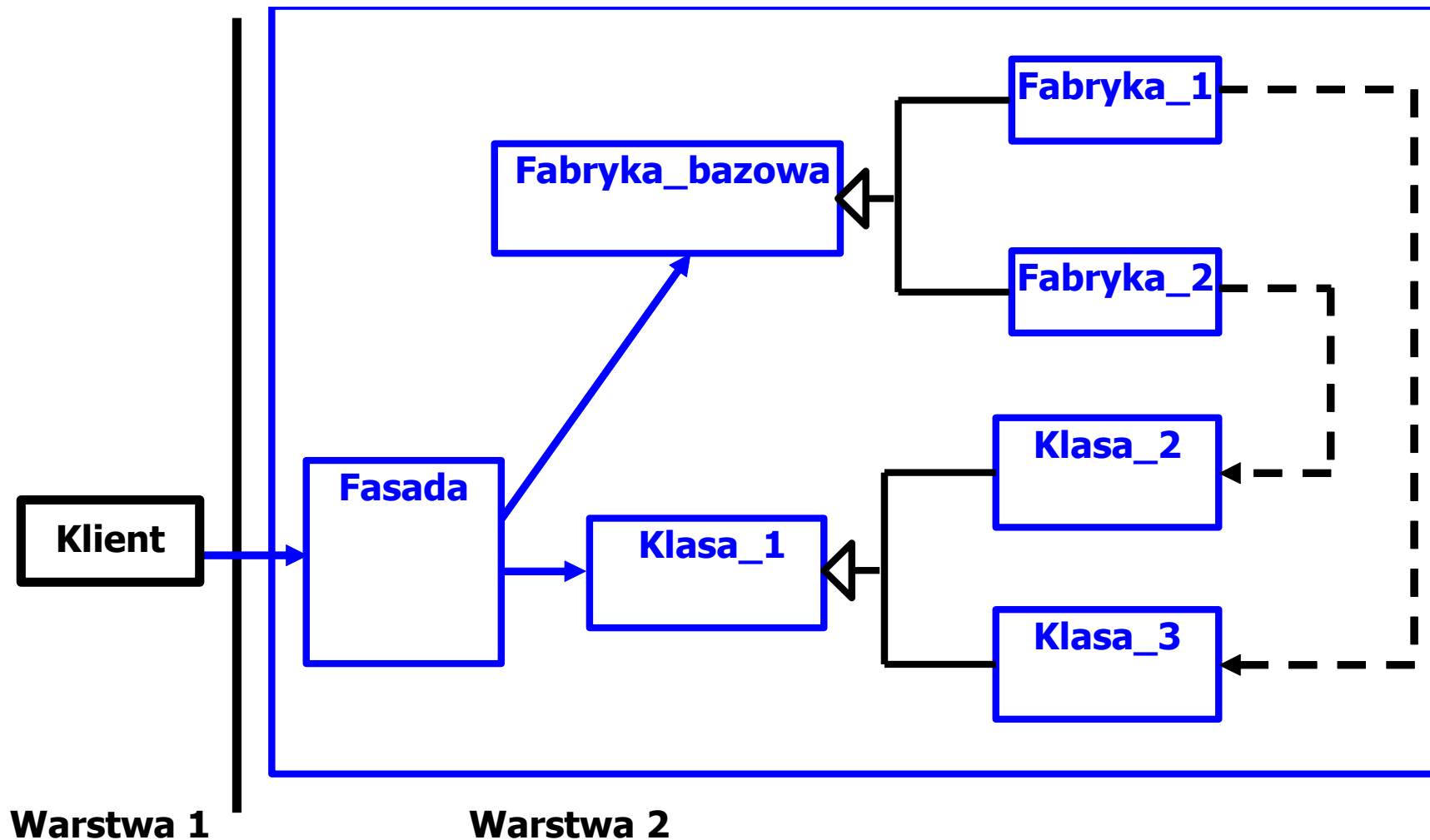
Identyfikacja wzorców projektowych (wykład 5 – część 2)

- Dobrze zbudowany system obiektowy jest pełen wzorców obiektowych
- Wzorzec to zwyczajowo przyjęte rozwiązanie typowego problemu w danym kontekście
- Strukturę wzorca przedstawia się w postaci diagramu klas
- Zachowanie się wzorca przedstawia się za pomocą diagramu sekwencji
- **Wzorce projektowe: Wzorzec reprezentuje powiązanie problemu z rozwiązaniem**

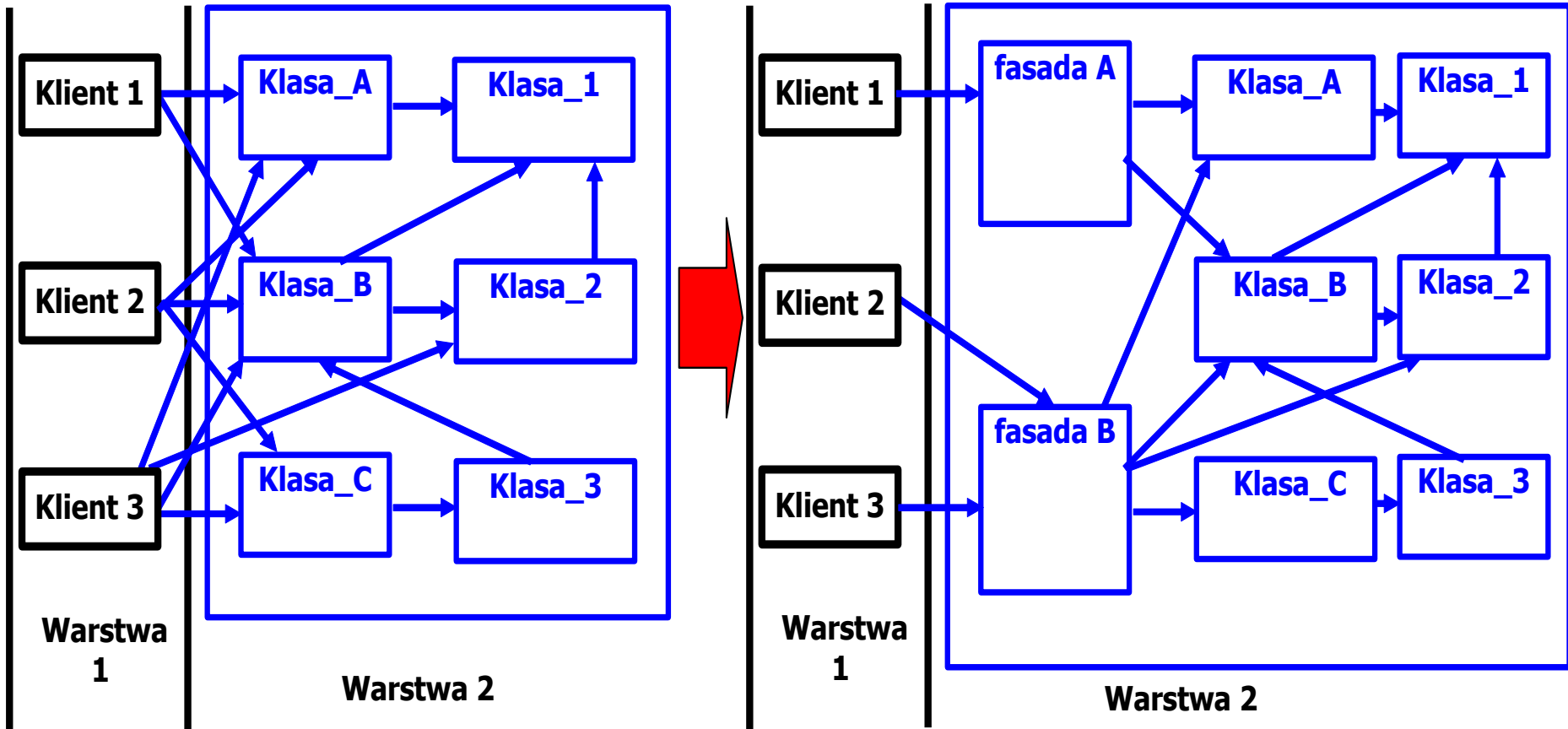
(wg Booch G., Rumbaugh J., Jacobson I., UML przewodnik użytkownika)

- Każdy wzorzec składa się z trzech części, które wyrażają związek między konkretnym kontekstem, problemem i rozwiązaniem (**Christopher Aleksander**)
- Każdy wzorzec to trzyczęściowa reguła, która wyraża związek między konkretnym kontekstem, rozkładem sił powtarzającym się w tym kontekście i konfiguracją oprogramowania pozwalającą na wzajemne zrównoważenie się tych sił w celu rozwiązania zadania. (**Richar Gabriel**)
- **Wzorzec to pomysł, który okazał się użyteczny w jednym rzeczywistym kontekście i prawdopodobnie będzie użyteczny w innym.** (**Martin Fowler**)

3.1. Wzorzec uniwersalny kreacyjny stosowany w każdej z warstw:
Fabryka obiektów (wykład 5 – część 2) –
oddzielenie tworzenia obiektów od zarządzania nimi i używania ich

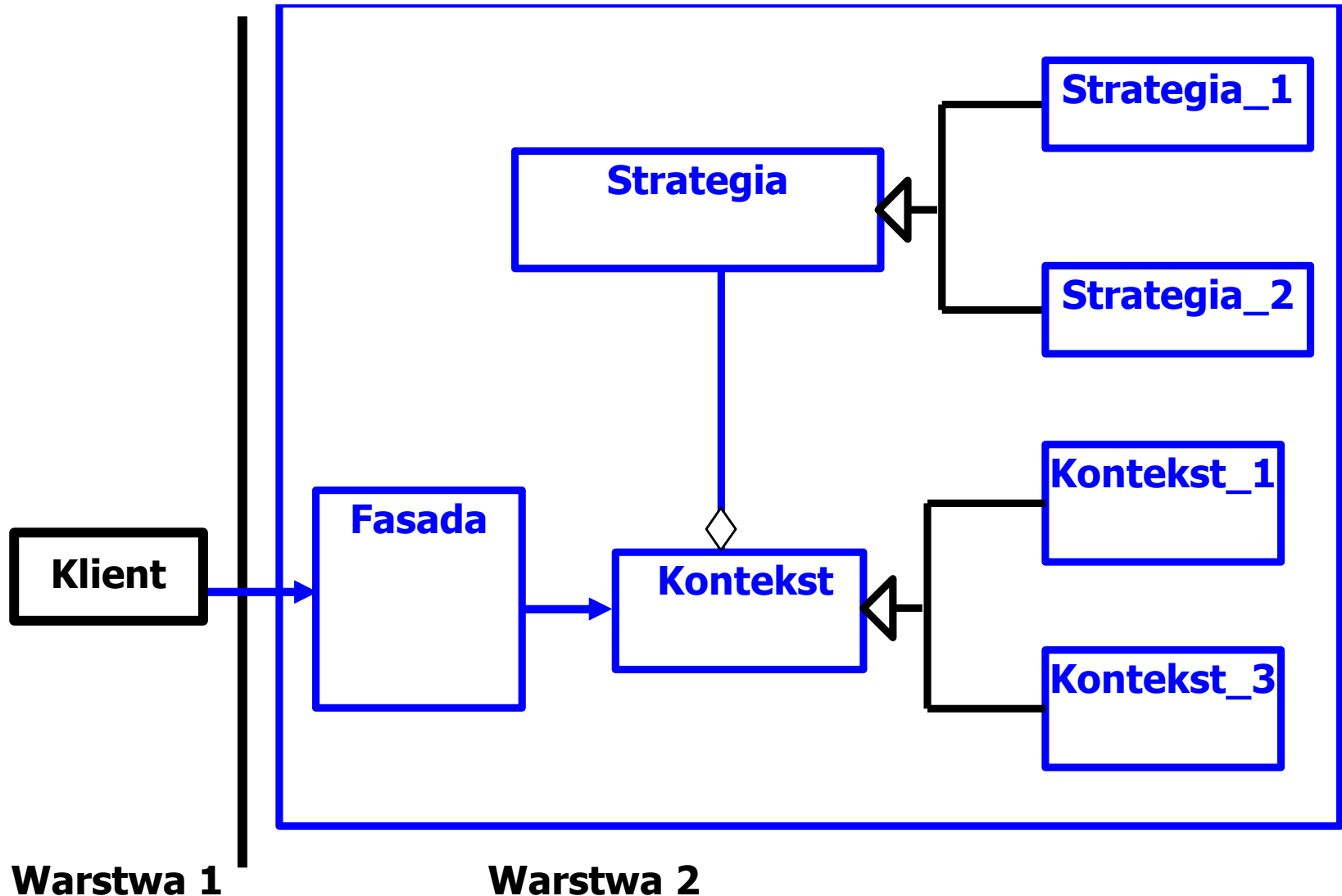


3.2. Wzorzec uniwersalny strukturalny: *Fasada* (wykład 5 – część 2) – hermetyzacja logiki biznesowej

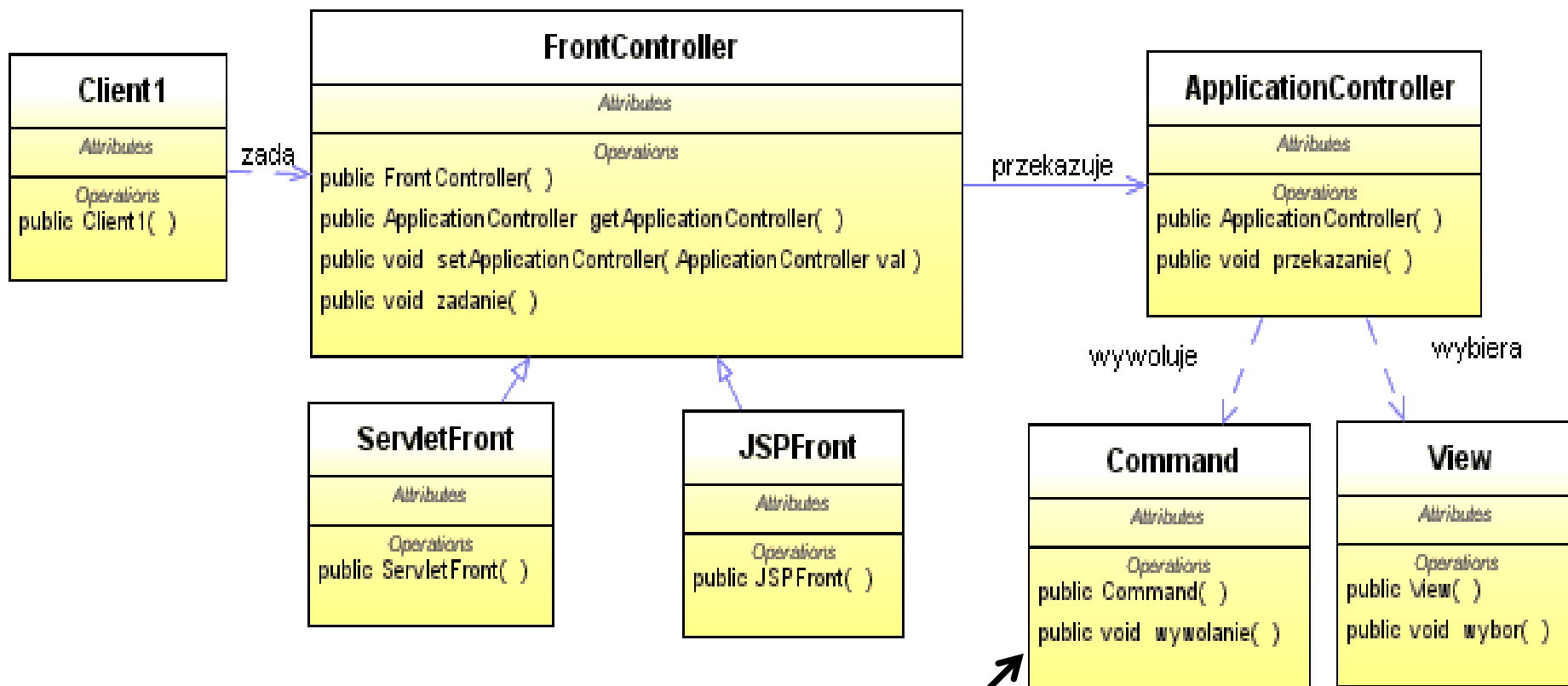


3.3. Wzorzec uniwersalny czynnościowy kreacyjny stosowany w każdej z warstw: : *Strategia* (wykład 5 – część 2)

– zastosowanie polimorfizmu do wyboru algorytmu



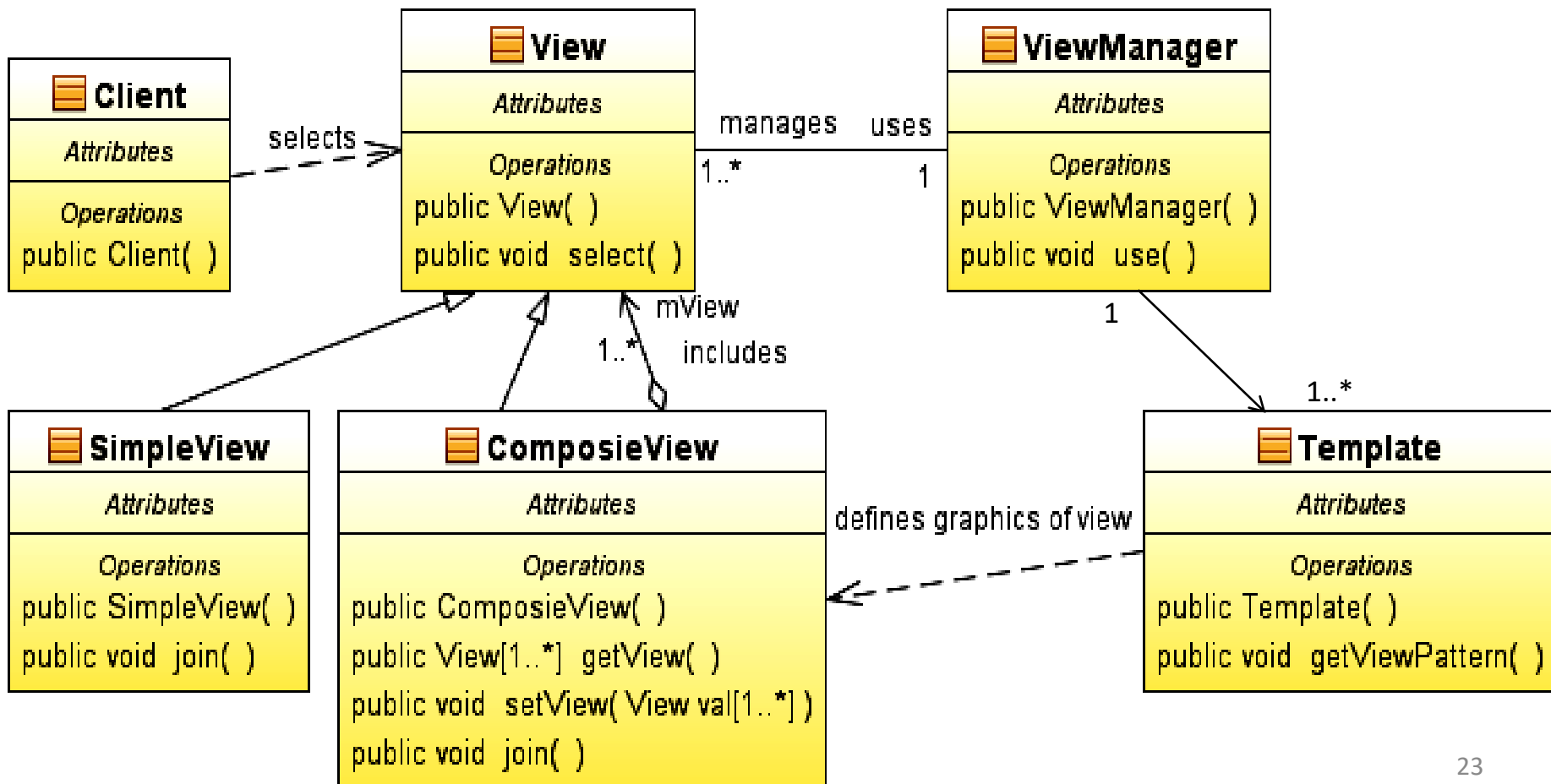
3.4. Wzorzec **EE Warstwy prezentacji: FrontController** – scentralizowany punkt dostępowy do obsługi żądań w *Warstwie prezentacji*



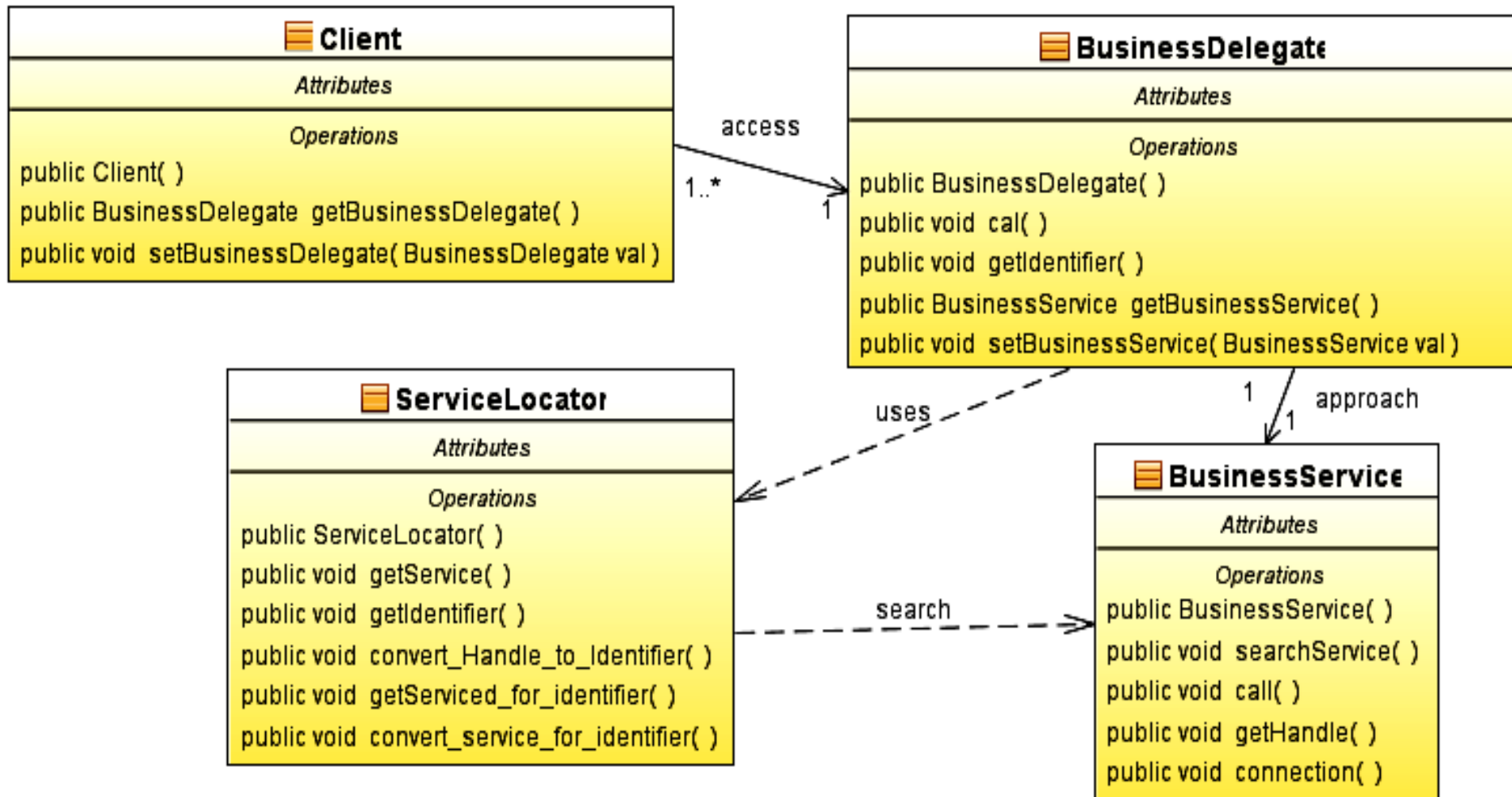
Wywołanie metod z *Warstwy biznesowej* np. za pośrednictwem wzorca **ApplicationService** (wzorzec *Warstwy biznesowej*)

3.5. Wzorzec **EE Warstwy prezentacji** : **Composite View** - widok kompozytowy powinien mieć strukturę modułową, zbudowaną z komponentów prostych, które razem tworzą złożoną stronę są zarządzane niezależnie

(**Client** == wzorzec **Warstwy prezentacji ApplicationController**).



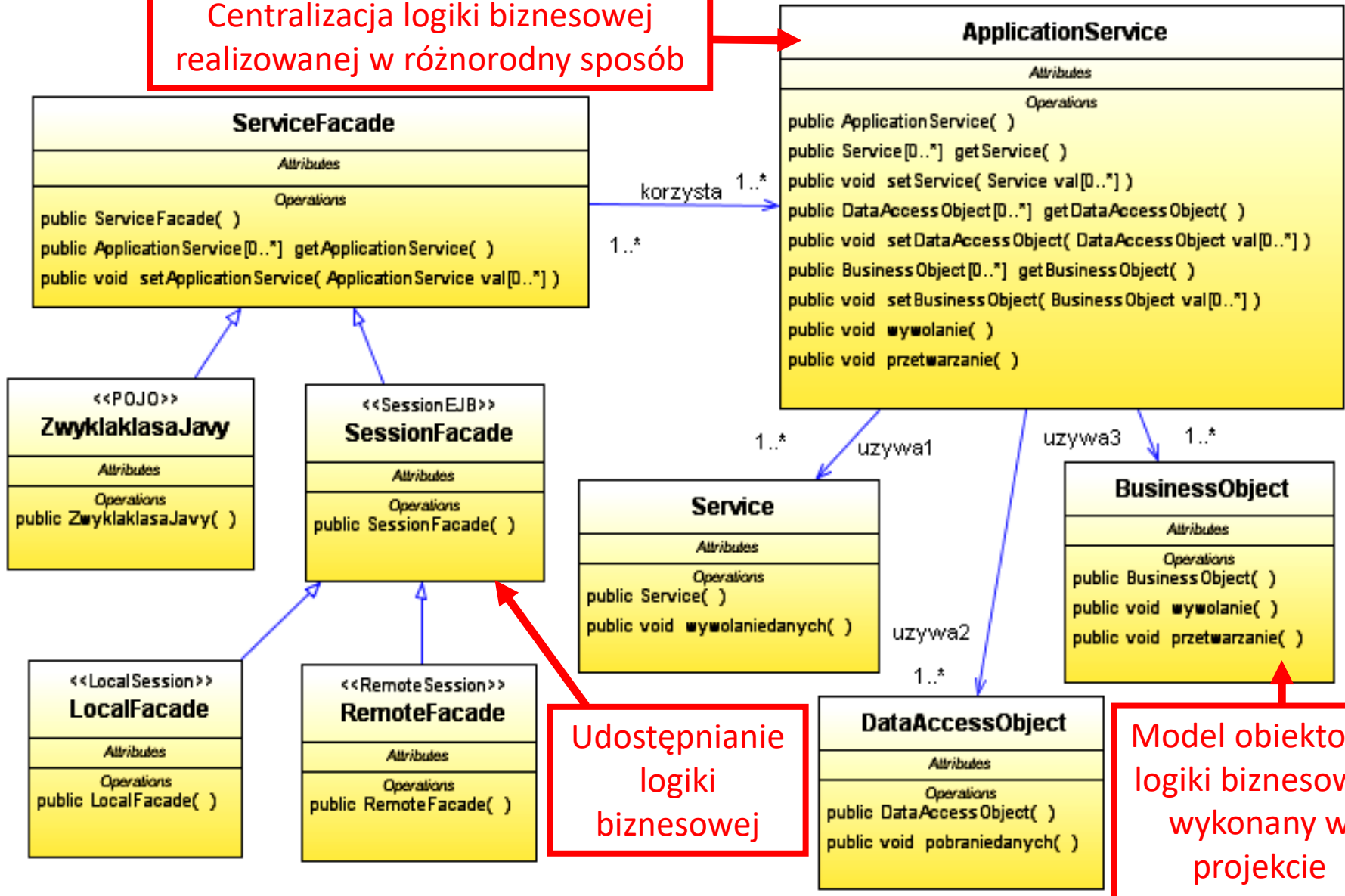
3.6. Wzorzec **EE Warstwy prezentacji**: do zdalnego wywołania usług z **Warstwy klienta** w celu ukrycia złożoności zdalnej komunikacji z komponentem usług biznesowych - **BusinessDelegate** (**Client**== wzorzec **Warstwy prezentacji ApplicationController**)



Komponenty EJB, JMS , lub część wzorca *SessionFacade*

3.7. Wzorce **EE Warstwy biznesowej: SessionFacade, ApplicationService** – udostępnianie i centralizacja logiki biznesowej kilku komponentów i usług biznesowych

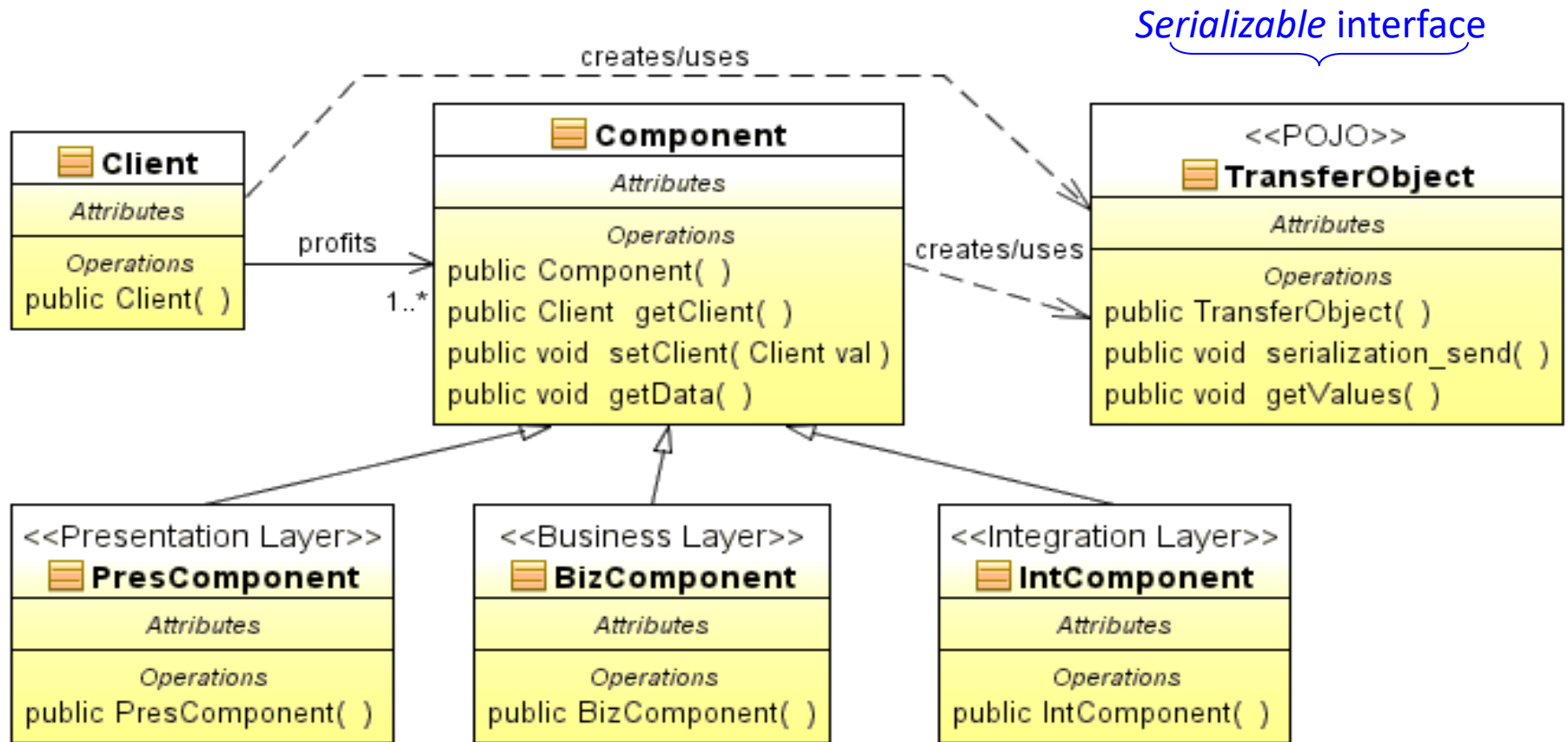
Centralizacja logiki biznesowej realizowanej w różnorodny sposób



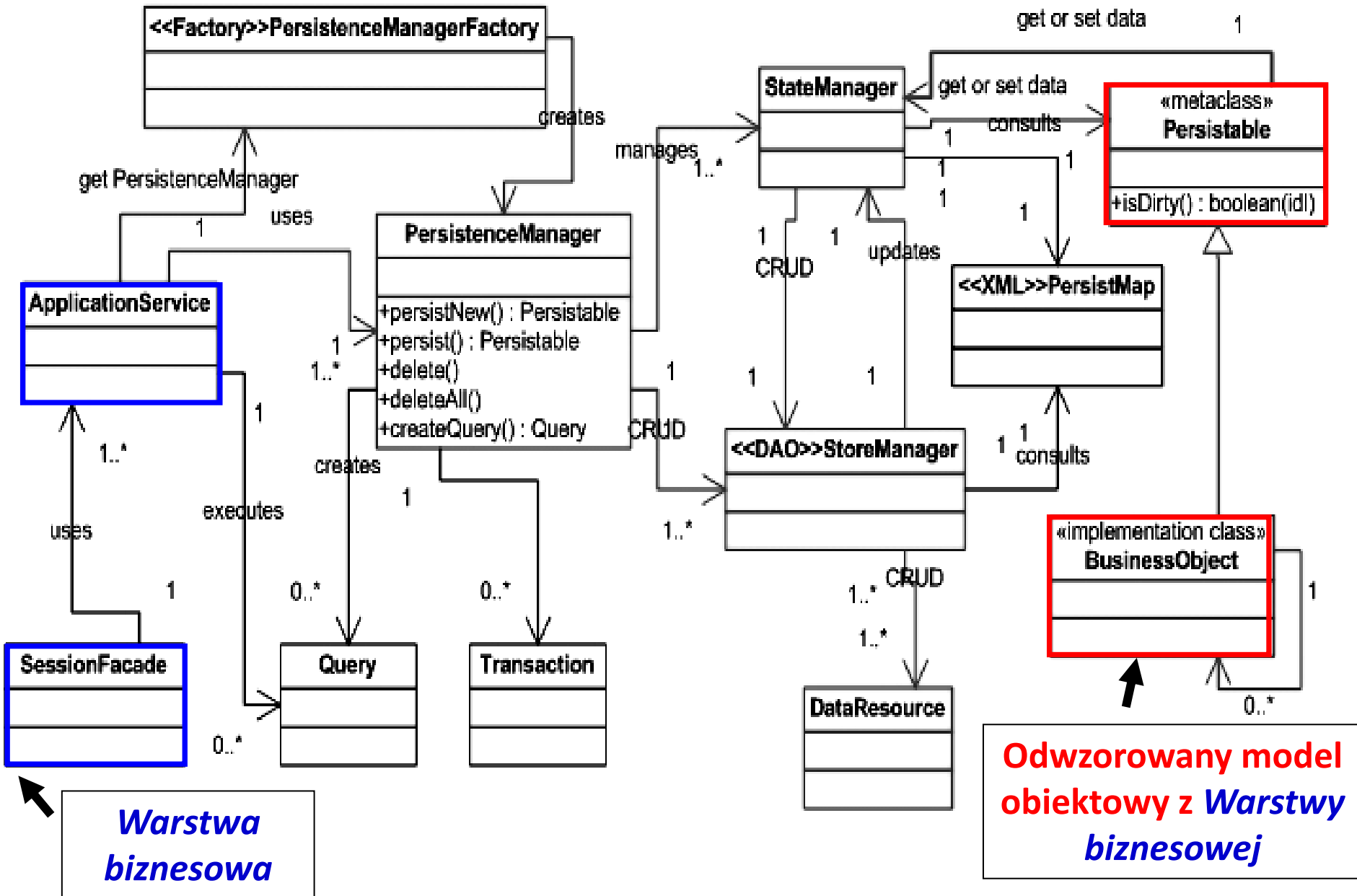
Udostępnianie logiki biznesowej

Model obiektowy logiki biznesowej wykonany w projekcie

3.8. Wzorce **EE** *Warstwy prezentacji, biznesowej, integracji*: **TransferObject** - przesyłanie danych między warstwami aplikacji (zmniejszanie ruchu w sieci poprzez zmniejszanie liczby połączeń zdalnych lub zwiększanie wydajności oraz zapewnienie hermetyzacji warstw)



3.9. Wzorzec **EE Warstwy integracji: DomainStore (ORM)** – oddzielenie mechanizmów trwałości od modelu obiektowego



Wyniki eksperymentów dostępu do baz danych z wykorzystaniem wzorców **DAO (JDBC) i puli połączeń oraz wzorców projektowych **Domains Store****

Requests no.	Jdbc[ms] (DAO)	Jndi[ms] (DAO and pool of connections)	Orm[ms] (Domain Store)
			lazy
20	8 431	5 136	2 627
100	49 356	19 311	2 445

Zagadnienia

1. **Wielowarstwowa architektura systemu informatycznego**
2. **Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury**
3. **Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej**
4. **5 zasad programowania solidnego (solid) [5]**

1. Zasada pojedynczej odpowiedzialności (Single-Responsibility Principle – SRP)

Żadna klasa nie może być modyfikowana z więcej niż jednego powodu.

1) Klasa obsługująca reguły biznesowe nie powinna zarządzać trwałością

2) Klasa tworząca obiekty nie powinna ich używać

3) Oddzielanie wzajemnie powiązanych odpowiedzialności- np. obiektowa idea systemu sporządzania rachunków:

–Zmiana sposobu wyznaczania ceny produktu: z podatkiem lub bez podatku - tylko modyfikacja kodu klas z rodziny

ProduktZPodatkiem, ProduktBezPodatku

–Zmiana numeru rachunku – ściśle związany z klasą **Rachunek**

–Obiekt typu **Promocja** powiązany z obiektem typu **ProduktBezPodatku** określa, czy należy zmienić cenę w obiektach z rodziny **ProduktZPodatkiem, ProduktBezPodatku**

2. Zasada otwarte-zamknięte (Open/Closed Principle – OCP)

Składniki oprogramowania (klasy, moduły, funkcje itp.) powinny być otwarte na rozbudowę, ale zamknięte dla modyfikacji .

- 1) Stosowanie **dziedziczenia, polimorfizmu, implementacji interfejsów** tylko w takich przypadkach, gdy istnieje możliwość zmian.
- 2) Należy wyeliminować rozpoznawanie klas zarządzanych lub używanych np. instrukcją **switch** przez klasy, które **używają** lub **zarządzają** tymi klasami
- 3) Przykłady: obiektowa idea zbioru produktów:
 - Zmiana strategii wyznaczania ceny produktu - tylko modyfikacja kodu klas przez polimorfizm i dziedziczenie: tylko klasa **ProduktZPodatkiem**
 - Zmiana sposobu określania ceny produktu wynikającej z promocji – określa obiekt typu **Promocja**, powiązany z obiektem typu **ProduktBezPodatku** i przez dziedziczenie z obiektem typu **ProduktZPodatkiem**. Klasa ta może być rozbudowana przez dziedziczenie, zachowując polimorfizm metody **obliczCzescBruttoCeny**.

3. Zasada podstawiania **Liskov**

Musi istnieć możliwość zastępowania typów bazowych ich podtypami. Jest to warunek zasady otwarte-zamknięte (OCP).

- 1) Klasa potomna nie może mieć mniejszej funkcjonalności niż jej klasa bazowa. Podstawianie klasy potomnej powinno następować automatycznie, bez potrzeby rozpoznawania typu obiektu np. instrukcją **switch**
- 2) Przykład: obiektowa idea systemu sporządzania rachunków:
 - automatyczne dostosowanie się do sposobu oznaczania typu produktu (klasy: **ProduktZPodatkiem**, **ProduktBezPodatku**)
 - automatyczne dostosowanie się do typu obiektu z rodziny **ProduktBezPodatku** zwracanego przez metodę **wykonajProdukt** obiektu typu **Fabryka**.

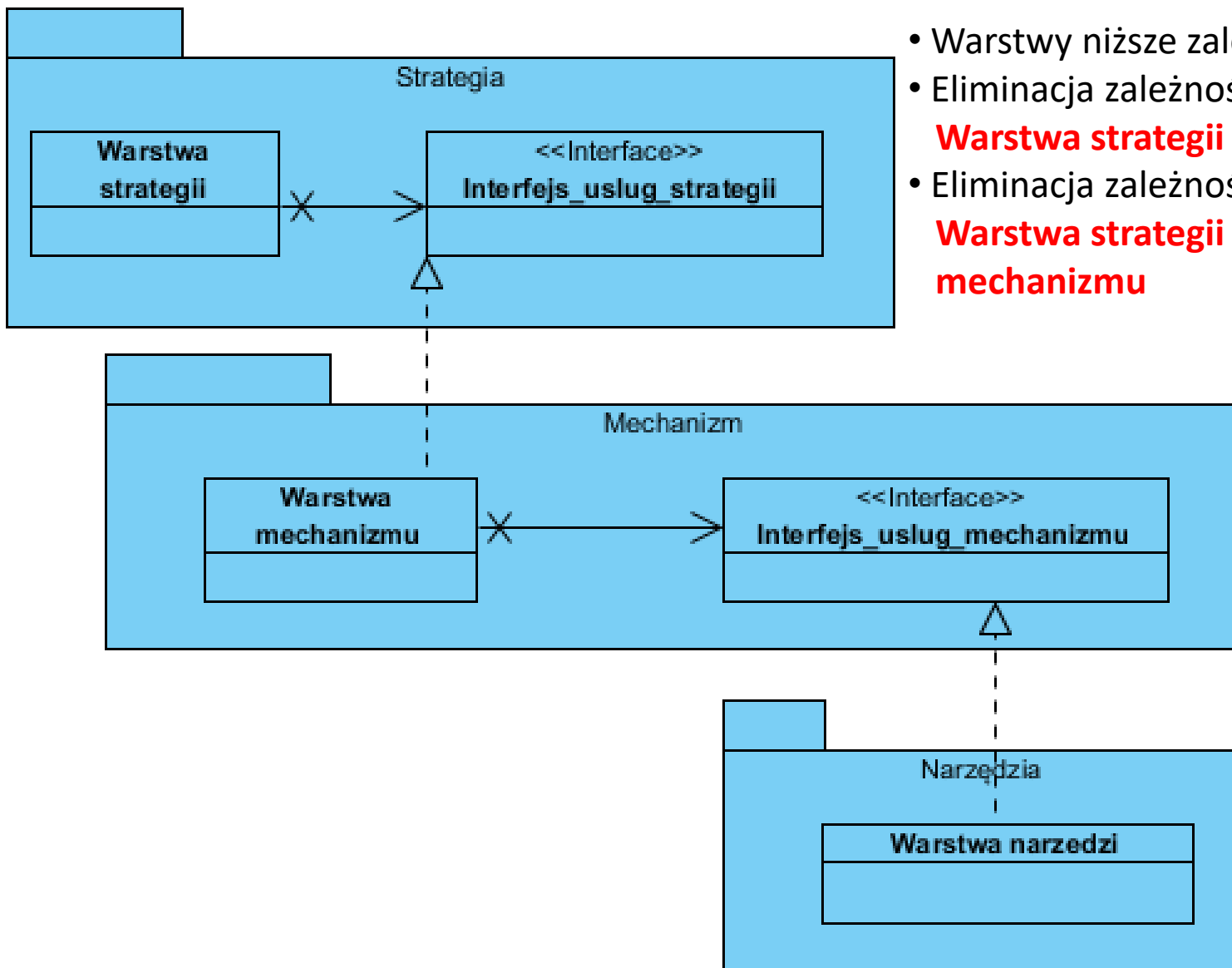
4. Zasada odwracania zależności (**Dependency Inversion Principle –DIP**)

A. Moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. Obie grupy modułów powinny zależeć od abstrakcji

B. Abstrakcje nie powinny zależeć od szczegółowych rozwiązań. To szczegółowe rozwiązania powinny zależeć od abstrakcji.

- 1. Strategia programu** nie powinna zależeć od szczegółowych rozwiązań w zakresie implementacji.
- 2. Interfejsy są związane ze swoimi właścicielami**, a nie implementującymi je klasami
- 3. Warstwa strategii** może być wielokrotnie używana w dowolnym kontekście pod warunkiem, że moduły niższego poziomu **implementują Interfejs_usług_strategii**

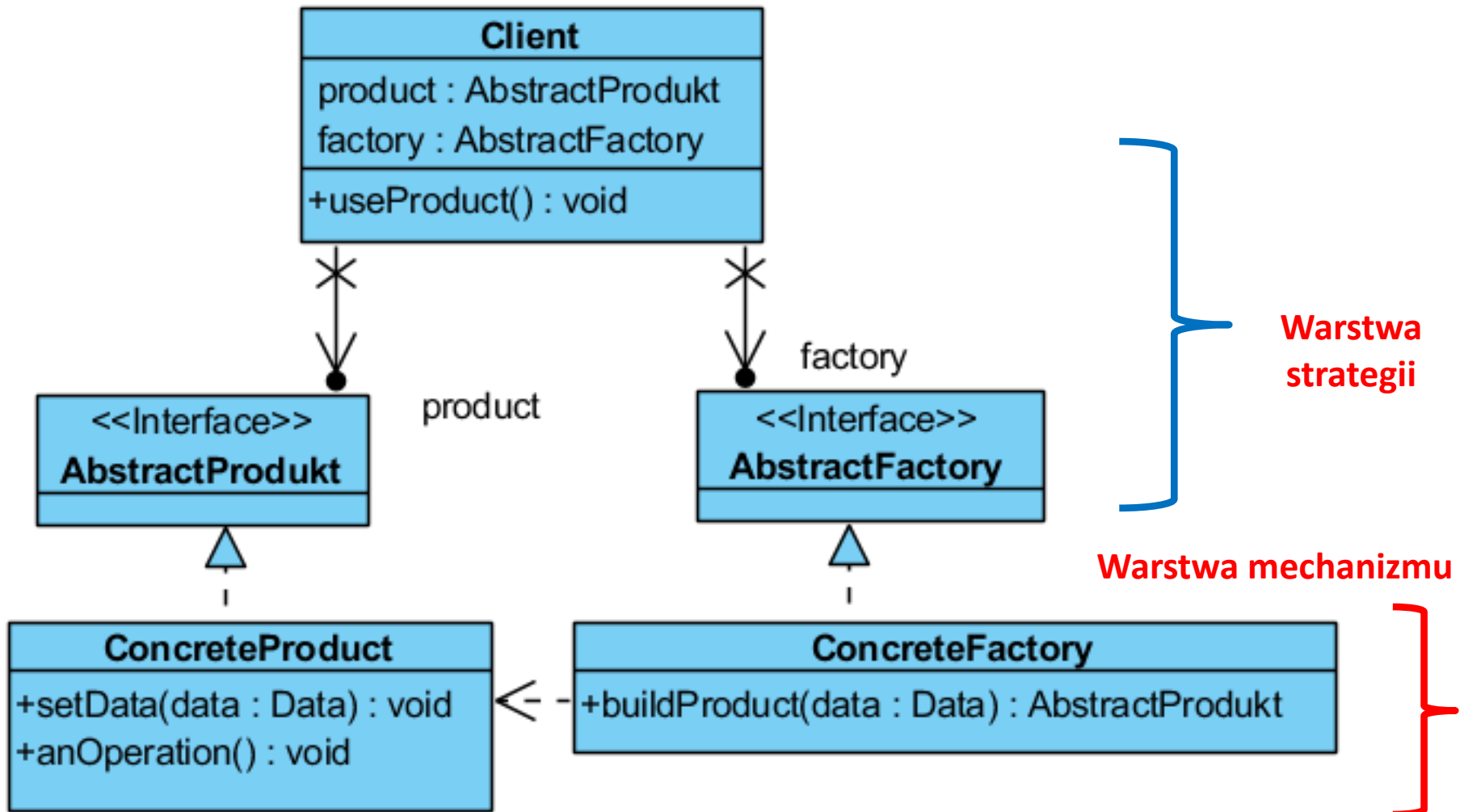
4 A - Podział oprogramowania na poziomy abstrakcji



- Warstwy niższe zależą od wyższych
- Eliminacja zależności przechodniej:
Warstwa strategii – warstwa narzędzi
- Eliminacja zależności bezpośredniej:
Warstwa strategii – **Warstwa mechanizmu**

4 B- Zasady procesu solidnego - przykład

Fabryka abstrakcyjna – *Abstract Factory*



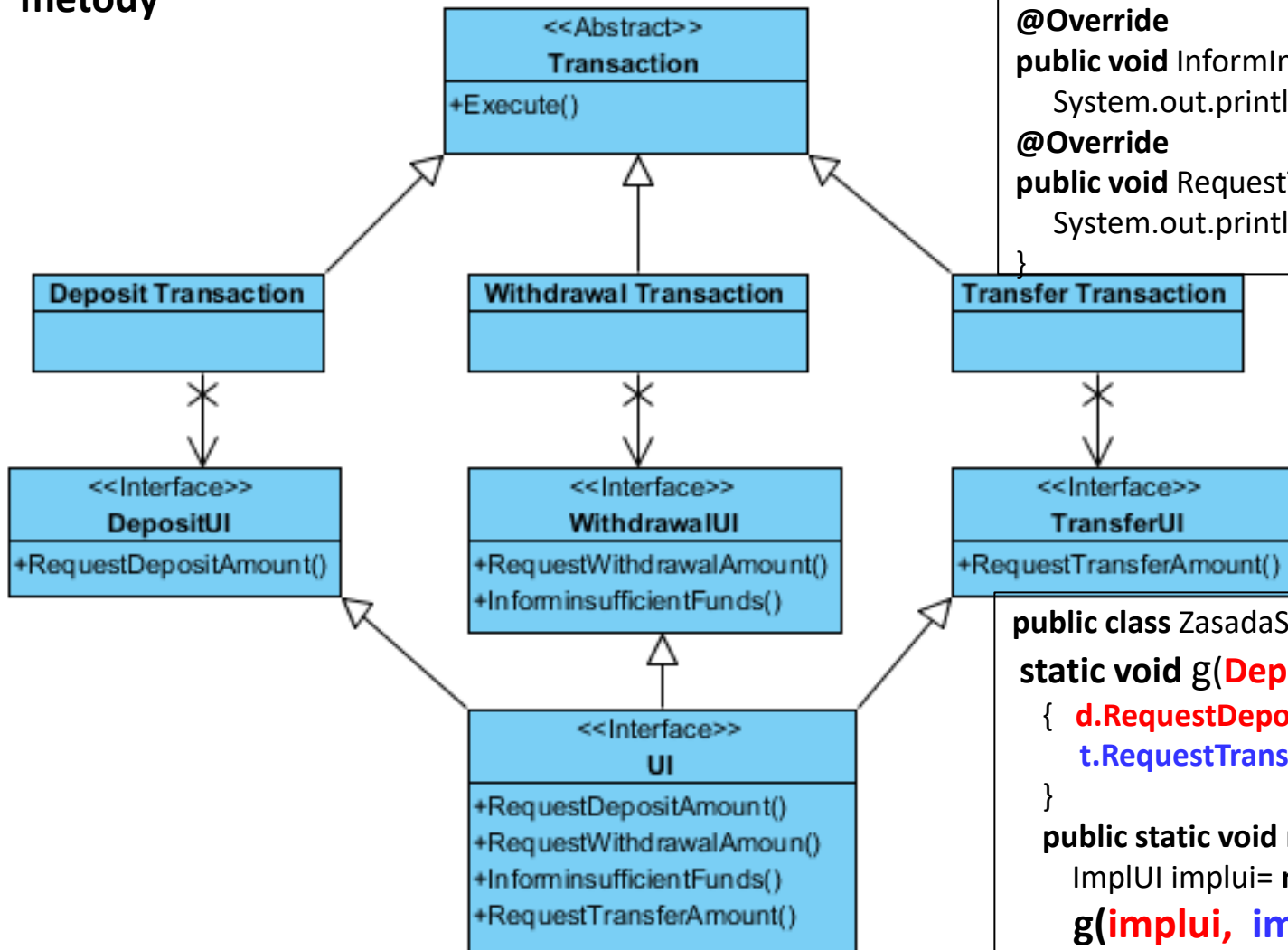
5. Zasada segregacji interfejsów (Interface Segregation Principle – ISP)

Klasa implementująca nie powinna być zmuszana do zależności od metod, których nie używa.

- 1) Separacja przez implementowanie wielu interfejsów
- 2) Dziedziczenie wielobazowe

Przykład ISP

- Trzy różne interfejsy xxxUI
- Możliwa jedna implementacja interfejsu UI →
- Każdy z interfejsów udostępnia jedynie swoje metody



```
public class ImplUI implements UI {  
    @Override  
    public void RequestDepositAmount() {  
        System.out.println("DepositRequest");  
    }  
    @Override  
    public void RequestWithdrawalAmount() {  
        System.out.println("WithdrawalRequest");  
    }  
    @Override  
    public void InformInsufficientFunds() {  
        System.out.println("WithdrawalInform");  
    }  
    @Override  
    public void RequestTransferAmount() {  
        System.out.println(("TransferRequest"));  
    }  
}
```

```
run:  
DepositRequest  
TransferRequest  
BUILD SUCCESSFUL
```

```
public class ZasadaSolid5 {  
    static void g(DepositUI d, TransferUI t)  
    {  
        d.RequestDepositAmount();  
        t.RequestTransferAmount();  
    }  
    public static void main(String[] args) {  
        ImplUI implui= new ImplUI();  
        g(implui, implui);  
    }  
}
```

Podsumowanie zasad programowania solidnego [5]

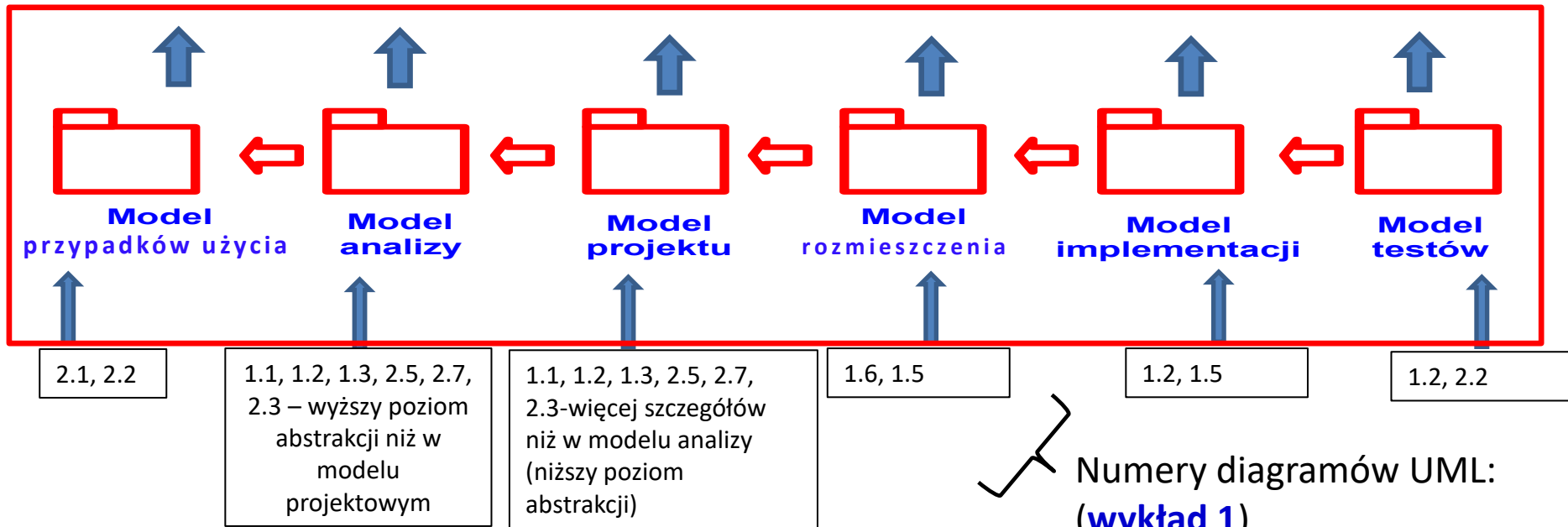
- Kluczowy mechanizm niskiego poziomu
- Podwyższa odporność kodu na zmiany
- Prowadzi do tworzenia kodu wielokrotnego użycia

Zagadnienia

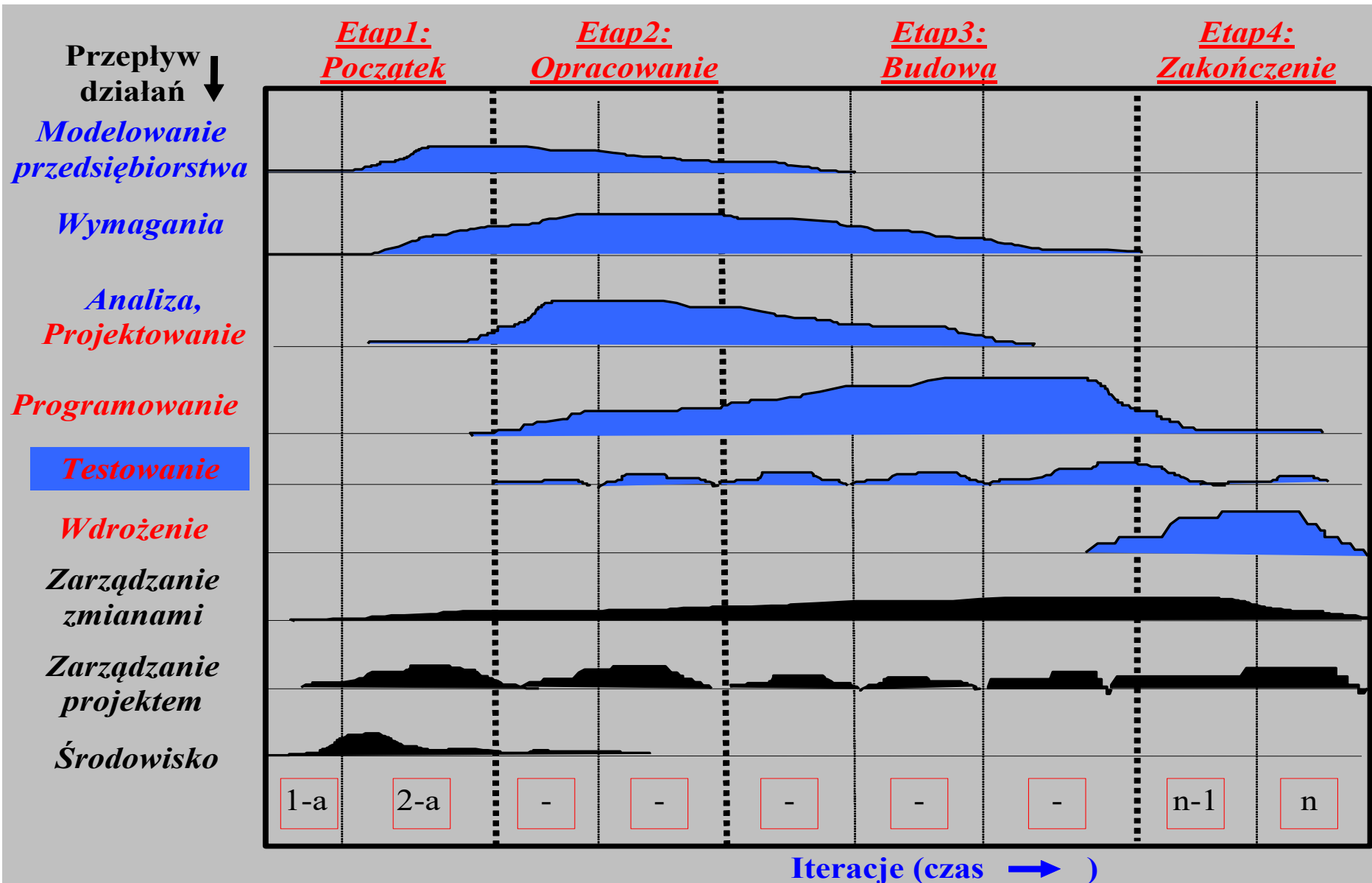
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu POJO. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.

Produkt - diagramy UML – modele, proces (wykład 1)

Modelowanie struktury i dynamiki systemu	Implementacja systemu,	struktury i dynamiki generowanie kodu
Perspektywa koncepcji <i>co należy wykonać?</i>	Perspektywa specyfikacji <i>jak należy używać?</i>	Perspektywa implementacji <i>jak należy wykonać?</i>
<ul style="list-style-type: none"> • model problemu np. przedsiębiorstwa • <u>wymagania</u> • <u>analiza</u> (model konceptualny: diagram przypadków użycia, diagram klas, diagramy sekwencji,) • <u>testy modelu</u> 	<ul style="list-style-type: none"> • <u>projektowanie</u> (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • <u>testy projektu</u> 	<ul style="list-style-type: none"> • <u>programowanie, wdrażanie</u> (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • <u>testy oprogramowania</u> • <u>wdrażanie</u> • <u>testy wdrażania</u>



Proces - zunifikowany iteracyjno- przyrostowy proces tworzenia oprogramowania – **kiedy należy wykonać?** [3LU] (wykład 1)



System informacyjny „Biblioteka” – przykład 3 z wykładu 2

Tworzenie modelu projektowego oraz implementacji

- I. Opis biznesowy „świata rzeczywistego”
- II. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych aplikacji
- III. Model przypadków użycia aplikacji oparty na diagramie przypadków użycia
- IV. Model analizy oparty na diagramie klas
- V. Model projektowy i implementacja **Warstwy biznesowej** oparty na diagramie klas i diagramie sekwencji tworzony metodą **iteracyjno-rozwojową**, sterowany realizacją przypadków użycia

I. Opis biznesowy „świata rzeczywistego” w języku klienta

1. Opis zasobów ludzkich

Co robią pracownicy?

2. Przepisy i strategia firmy

Co ogranicza działalność firmy?

3. Dane techniczne

Dane ilościowe:

ilu pracowników,
ile danych,
jak często,

Dane o lokalizacji firmy

Dane o klientach firmy

Dane o używanym sprzęcie i oprogramowaniu

Opis biznesowy „świata rzeczywistego” biblioteki.

1. Opis zasobów ludzkich

Bibliotekarz może **dodawać** do katalogu tytułów nowe tytuły. Każdy tytuł jest reprezentowany przez dane: tytuł, autor, wydawnictwo, ISBN oraz informacje o liczbie egzemplarzy i miejscu ich przechowywania i występuje w bibliotece jako pojedyncza informacja dla każdego tytułu. Pewna grupa tytułów opisuje książki nagrane na wybranym nosniku, dlatego dodatkowo tytuł zawiera dane nagrania np nazwisko aktora. Każdy egzemplarz, niezależnie, czy jest książką czy kasetą, jest opisany odrębną informacją zawierającą numer egzemplarza, który może się powtarzać dla różnych tytułów. Bibliotekarz może **dodawać** nowe tytuły i egzemplarze oraz je **przeszukiwać**, natomiast klient może jedynie przeszukiwać tytuły i sprawdzać egzemplarze wybranych tytułów.

W celu wypożyczenia książki **klient** musi ją **zarezerwować** podając dane rejestracji, dane książki oraz datę rezerwacji. Klient musi **wypożyczyć** zarezerwowaną książkę w terminie jej rezerwacji podając dane rejestracji i rezerwacji, wyszukać rezerwację i następnie ją usunąć. Musi wykonać dane wypożyczenia zawierające: dane rejestracji, dane zarezerwowanej książki oraz datę zwrotu. Rezerwacje można usunąć bez konieczności jej wypożyczenia – w terminie rezerwacji.

W celu **zwrotu książki** należy podać dane rejestracji oraz dane wypożyczonej książki. Zwrot musi nastąpić w okresie wypożyczenia podanym w danych wypożyczenia.

Opis biznesowy „świata rzeczywistego” biblioteki (cd)

2. Przepisy

Pracownik ponosi odpowiedzialność za poprawność danych - odpowiada materialnie za niezgodność danych ze stanem wypożyczalni.

3. Dane techniczne

Klient może przeglądać dane wypożyczalni za pośrednictwem strony internetowej lub bezpośrednio za pomocą specjalnego programu. Zakłada się, że klientów jednocześnie przeglądających dane wypożyczalni może być ponad 1000 oraz wypożyczalnia może zawierać kilkadziesiąt tysięcy tytułów oraz przynajmniej dwukrotnie więcej egzemplarzy. Biblioteka składa się z kilku ośrodków w różnych miastach na terenie kraju (lista miast jest dołączona do umowy). Zaleca się stosowanie technologii Java.

II. Sformułowanie wymagań funkcjonalnych i нефункциональных

Lista wymagań funkcjonalnych programu

1. System zawiera katalog tytułów
2. System zawiera dwa typy egzemplarzy do **rezerwacji** i następnie **wypożyczenia** i **zwrotu**: książki i kasety z nagraniami dźwiękowymi książek.
3. Każdy **dodawany** egzemplarz zawiera tytuł, nazwisko autora, ISBN, wydawnictwo, jeśli jest to książka oraz dodatkowo nazwisko aktora, jeżeli jest to nagranie dźwiękowe.
Może wystąpić wiele egzemplarzy książek oraz kaset z tymi samymi tytułami. Każdy egzemplarz, zarówno książka i kaseeta, posiadają numer niepowtarzający się w ramach pozostałych identycznych danych (ISBN lub ISBN i nazwisko aktora).
4. W celu **znalezienia** tytułu należy podać ISBN lub ISBN i nazwisko aktora, jeżeli należy odszukać tytuł nagranej książki.
5. W celu wybrania właściwego egzemplarza należy podać ISBN, jeśli jest to książka oraz dodatkowo nazwisko aktora, jeśli jest to kaseeta oraz numer egzemplarza.
6. Zarówno egzemplarze typu książka lub kaseeta, mogą być przeznaczane do wypożyczenia na okres umowny oraz na okres ściśle określony.

Lista wymagań нефункциональных programu

1. **Wstawianie** danych o tytułach i egzemplarzach może odbywać się tylko przez uprawnione osoby
2. **Wyszukiwanie**, **rezerwacja**, **wypożyczenie** i **zwrot** egzemplarzy powinny odbywać się samodzielnie przez klienta
3. Operacje zarządzania i wyszukiwania informacji mogą być dokonane przez Internet przez aplikację uruchamianą przez przeglądarkę lub bez jej pośrednictwa

Diagram wymagań funkcjonalnych programu

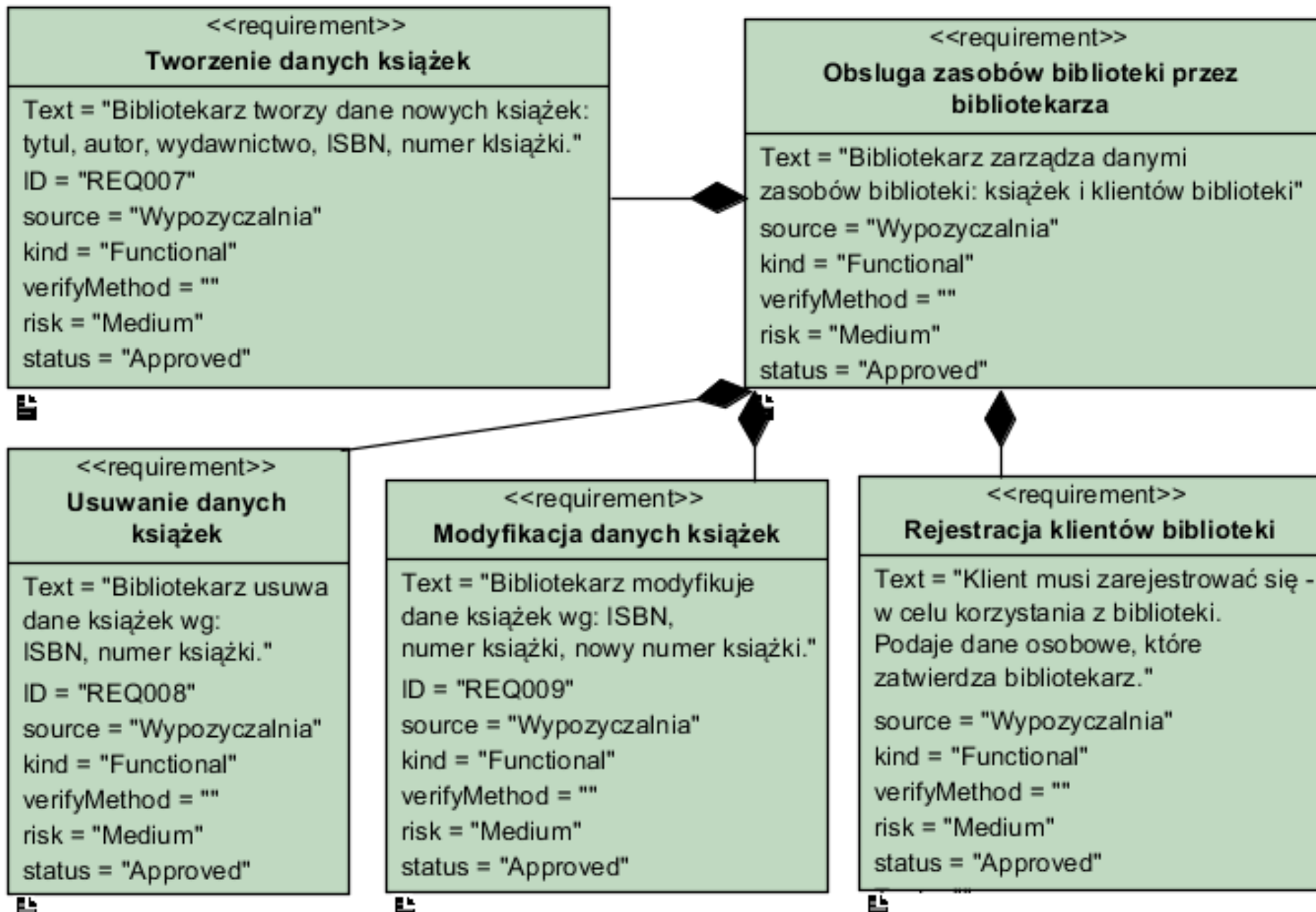


Diagram wymagań funkcjonalnych programu(cd)

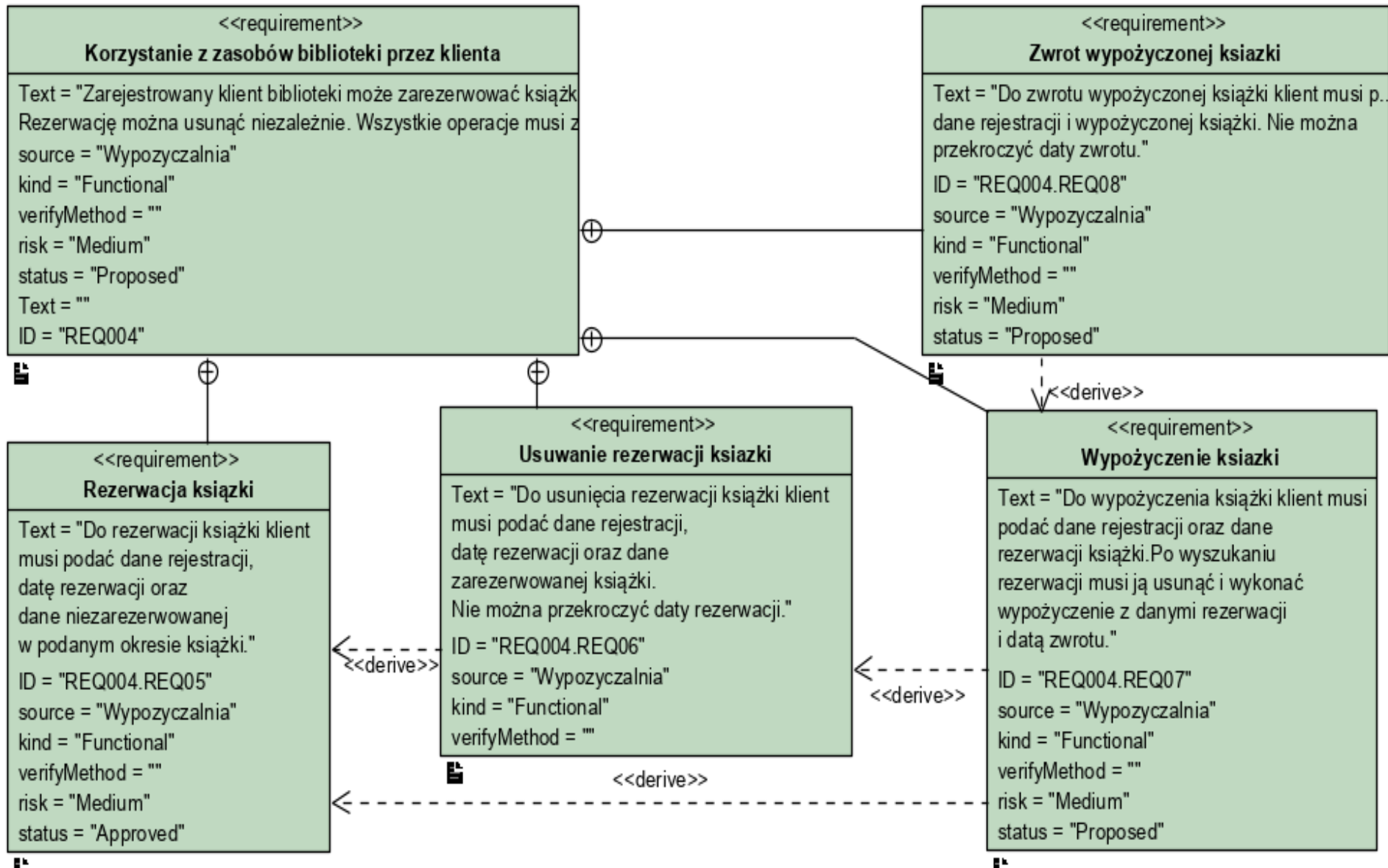
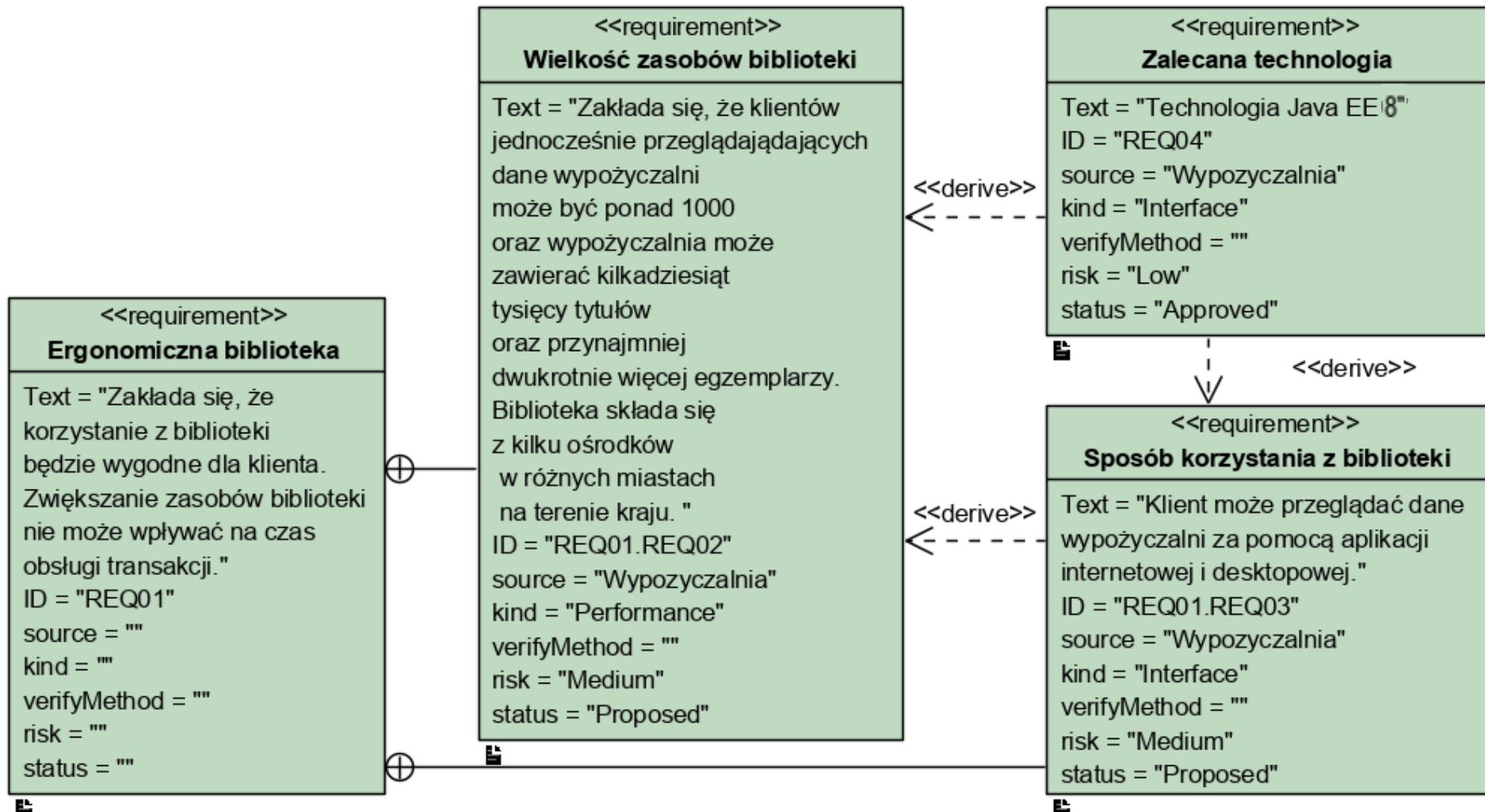


Diagram wymagań niefunkcjonalnych programu (cd)



III. Model analizy aplikacji oparty na diagramie przypadków użycia

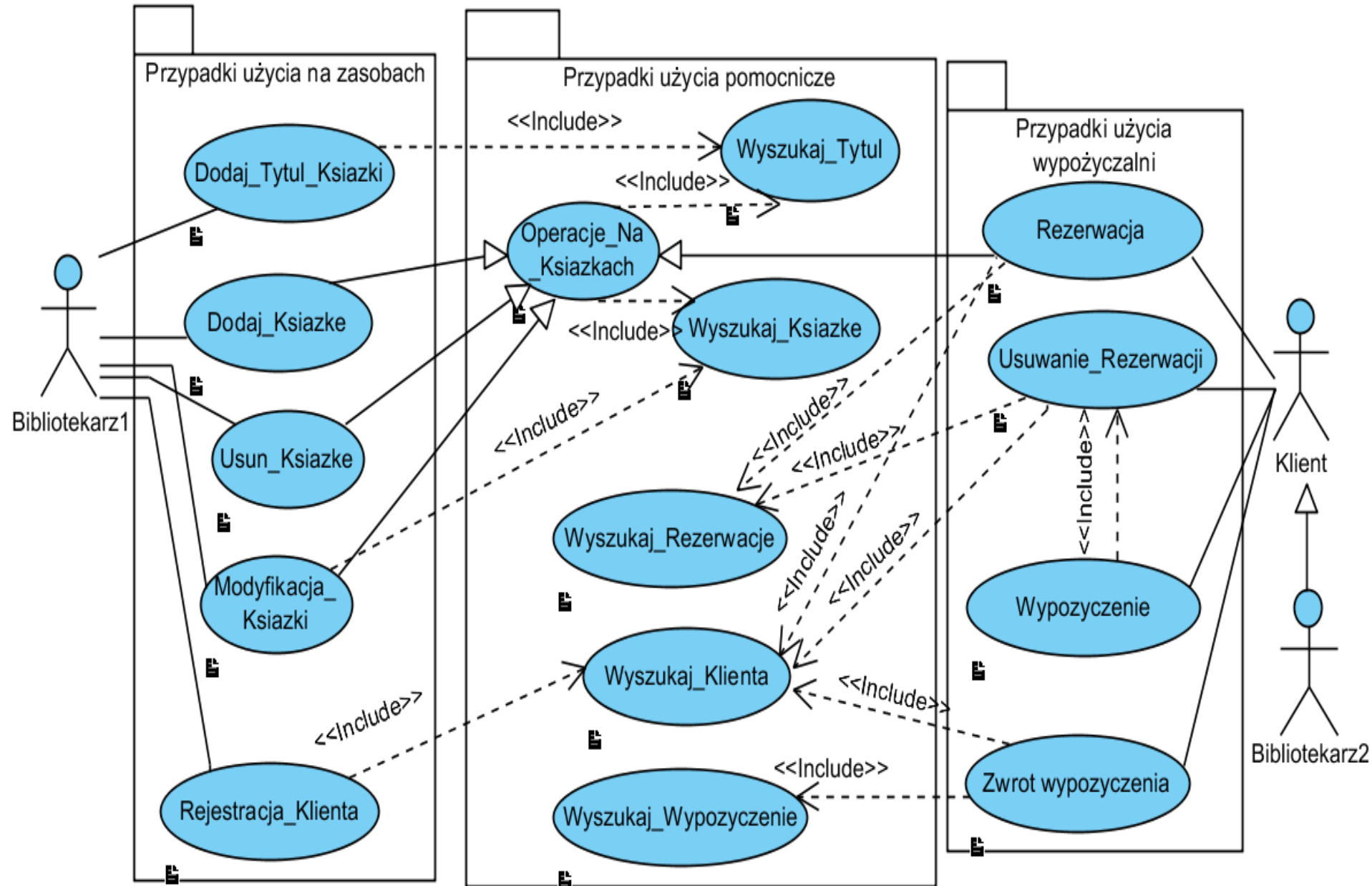
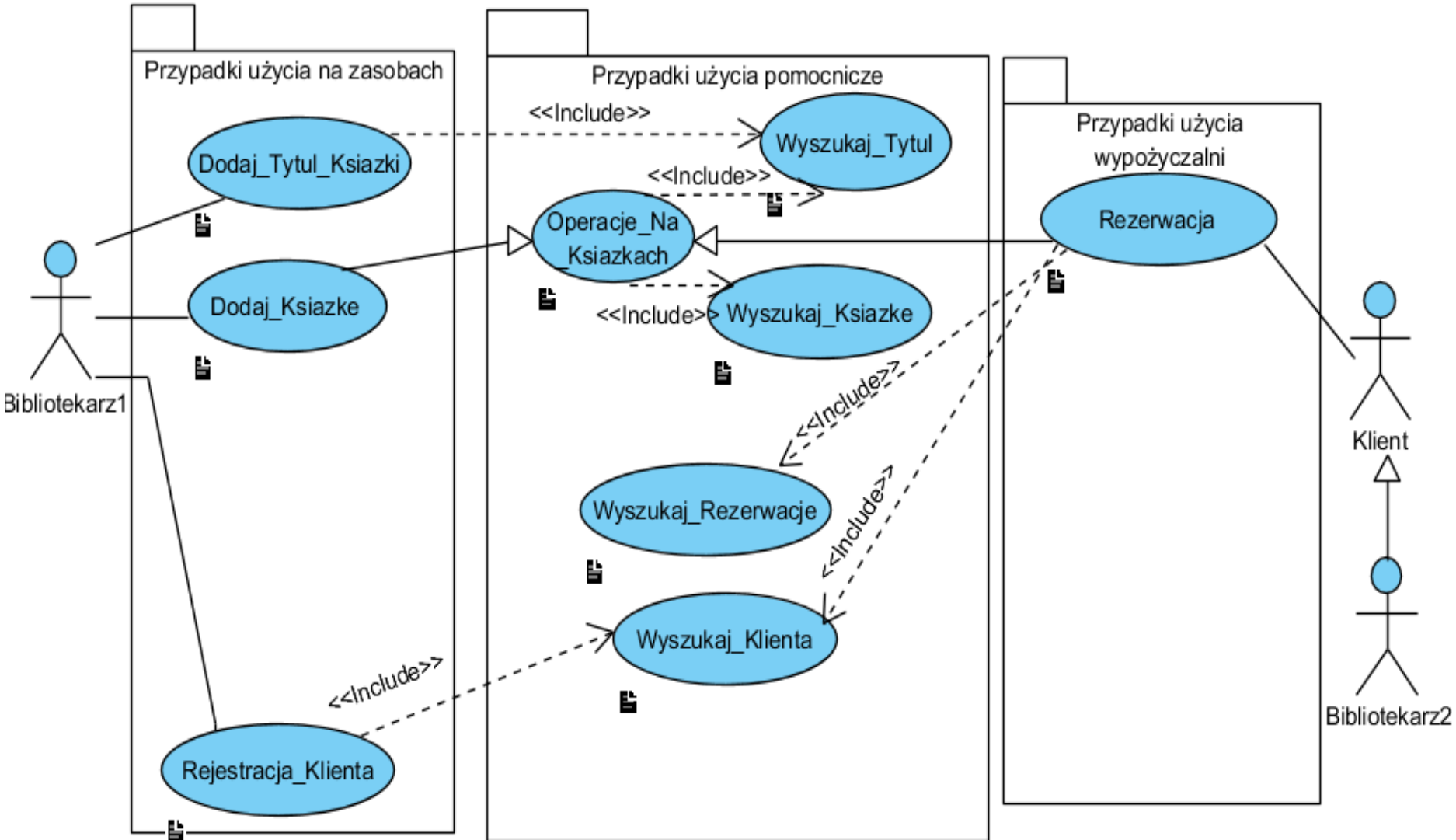


Diagram przypadków użycia - wybrany fragment bez realizacji wypożyczenia



IV. Model analizy *Warstwy biznesowej* oparty na diagramie klas utworzony na podstawie analizy przypadków użycia.

Podjęcie zaprezentowane w instrukcji do laboratorium 1 -

http://zofia.kruckiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/Instrukcja_1_2.pdf

Identyfikacja klas – etap 1

Analiza wspólności (perspektywa koncepcji, model analizy – wykład 1)

- Wykryto **cztery główne klasy** typu **Entity** ze względu na odpowiedzialność:
 - **TitleBook** (PU: **Dodaj_Tytul_Ksiazki, Dodaj_Ksiazke, Rezerwacja**),
 - **Book** (PU: **Dodaj_Ksiazke, Rezerwacja**),
 - **Client** (PU: **Rejestracja_Klienta, Rezerwacja**)
 - **Reservation** (PU: **Rezerwacja**)

Identyfikacja klas – etap 2

Analiza zmienności (perspektywa specyfikacji, model projektowy – wykład 1)

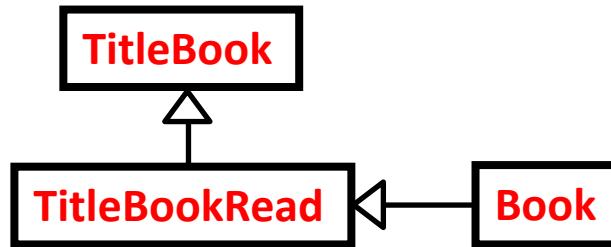
- Wykryto **dziedziczenie** w właściwościach tytułów książek, które określają, które książki są nagrane, a które są drukowane
Zdefiniowano klasę pochodną:
 - **TitleBookRead** typu **Entity**, która dziedziczy od klasy **TitleBook** i zawiera informację o aktorze, która czyta książkę na nagraniu (PU: **Dodaj_Tytul_Ksiazki**, **Dodaj_Ksiazke**, **Rezerwacja**)

Analiza wspólności i zmienności - identyfikacja typów relacji

Oszacowania dla przyjętego modelu powiązań:

- Liczba obiektów z typu **TitleBookRead** : 5000,
- Przybliżony największy rozmiar obiektu typu **TitleBookRead**: R1
- Przybliżony największy rozmiar obiektu typu **Book**: R2
- Średnia liczba książek na 1 obiekt z rodziny typu **TitleBookRead**: 50
- Liczba wszystkich książek: 250000

Oszacowania dla częściowo równoważnego modelu dziedziczenia:



- Liczba obiektów z rodziny **TitleBookRead**: 5000,
- Przybliżony największy rozmiar obiektu z typu **TitleBookRead**: R1
- Przybliżony największy rozmiar obiektu typu **Book**: R1+R2
- Średnia liczba książek na 1 obiekt z rodziny typu **TitleBook**: 50
- Liczba wszystkich książek: 250000

	Przyjęty model powiązań	Model oparty na dziedziczeniu
Rozmiar pamięci	$5000 * R1 + 5000 * 50 * R2$	$5000 * R1 + 5000 * 50 * (R1 + R2)$
Liczba przeszukań obiektów typu Book	od 1 do $(5000 + 50)$	Od 1 do $5000 * 50$

Analiza zmienności (c.d)

- **Wykryto związki:**

- asocjacji dwukierunkowej między obiektem typu **TitleBook** i obiektami typu **Book**: obiekt typu **TitleBook** zarządza zbiorem obiektów typu **Book**, a każdy obiekt typu **Book** jest powiązany tylko z jednym obiektem typu **TitleBook**, który określa jego tytuł oraz, czy jest książką nagrałą, czy też papierową (**PU Dodaj_Tytuł**, **PU Dodaj_Ksiazke**).
- asocjacji dwukierunkowej między klasą **Reservation** i klasami **Book** i **Client**. Obiekty typu **Client** i **Book** posiadają zbiór obiektów typu **Reservation**, a obiekt typu **Reservation** jest powiązany tylko z jednym obiektem typu **Client** i jednym obiektem typu **Book**
- Podstawą identyfikacji jest **PU Rezerwacja**

Analiza zmienności (c.d)

- Zastosowano wzorzec strukturalny typu Fasada:
 - klasę fasadową **Facade** typu **Control** do oddzielenia przetwarzania obiektów typu **Entity** od pozostałej części systemu
- Zastosowano wzorzec wytwórczy typu Fabryka
 - klasę typu **Control** jako fabrykę obiektów (**Factory**) do tworzenia różnych typów produktów – czyli obiektów typu **TitleBook**, **TitleBookRead**, **Book**, **Reservation**, **Client**

V. Model projektowy i implementacja **Warstwy biznesowej** oparty na diagramie klas i diagramie sekwencji tworzony metodą **iteracyjno-rozwojową**, sterowany **realizacją dwóch przypadków użycia**.
Podejście zaprezentowane w instrukcji do **laboratorium 1** -

http://zofia.kruckiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/Instrukcja_1_2.pdf

Iteracja 1

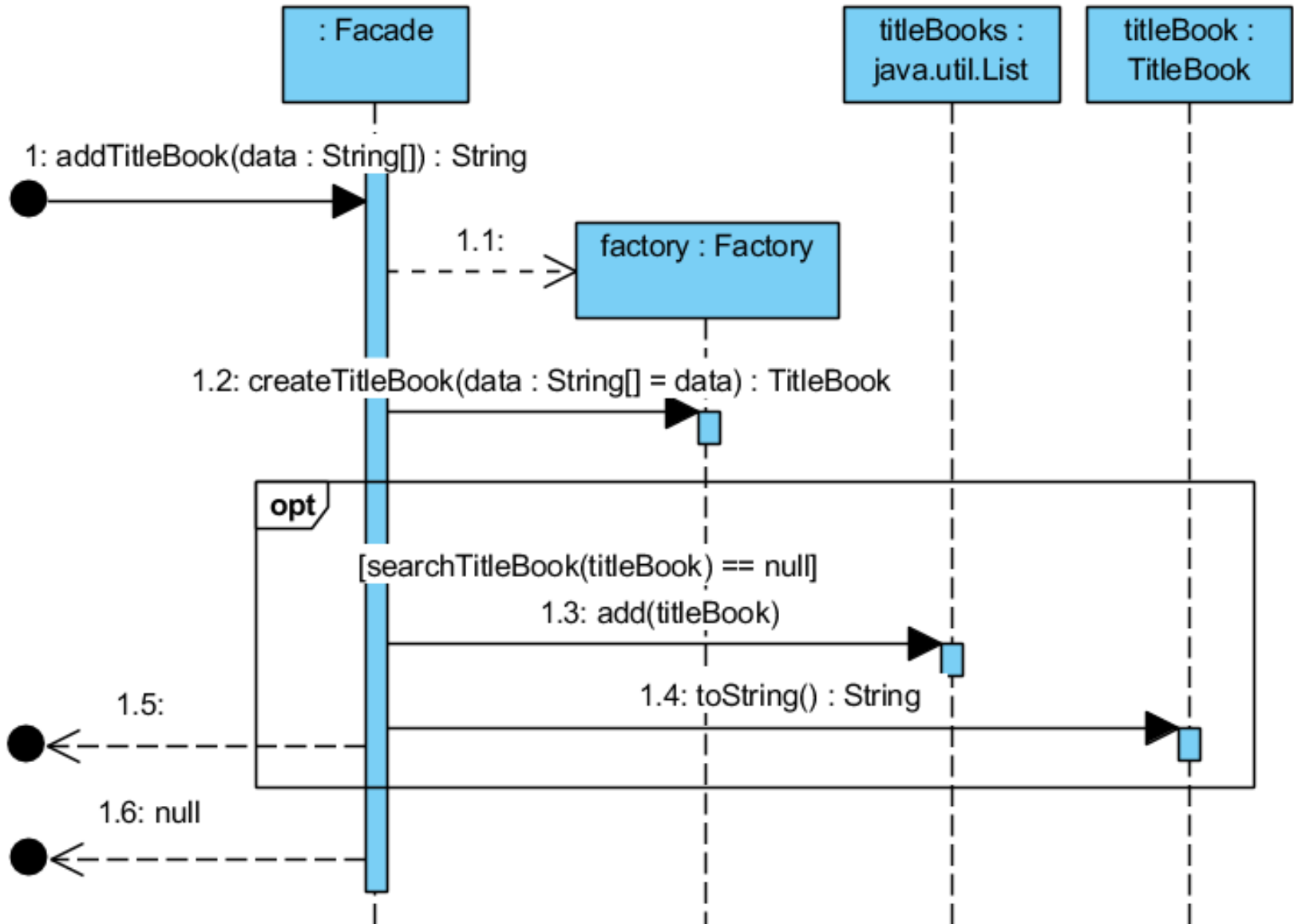
Projekt przypadku użycia

„ **Dodaj_Tytul_Ksiazki**”

za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

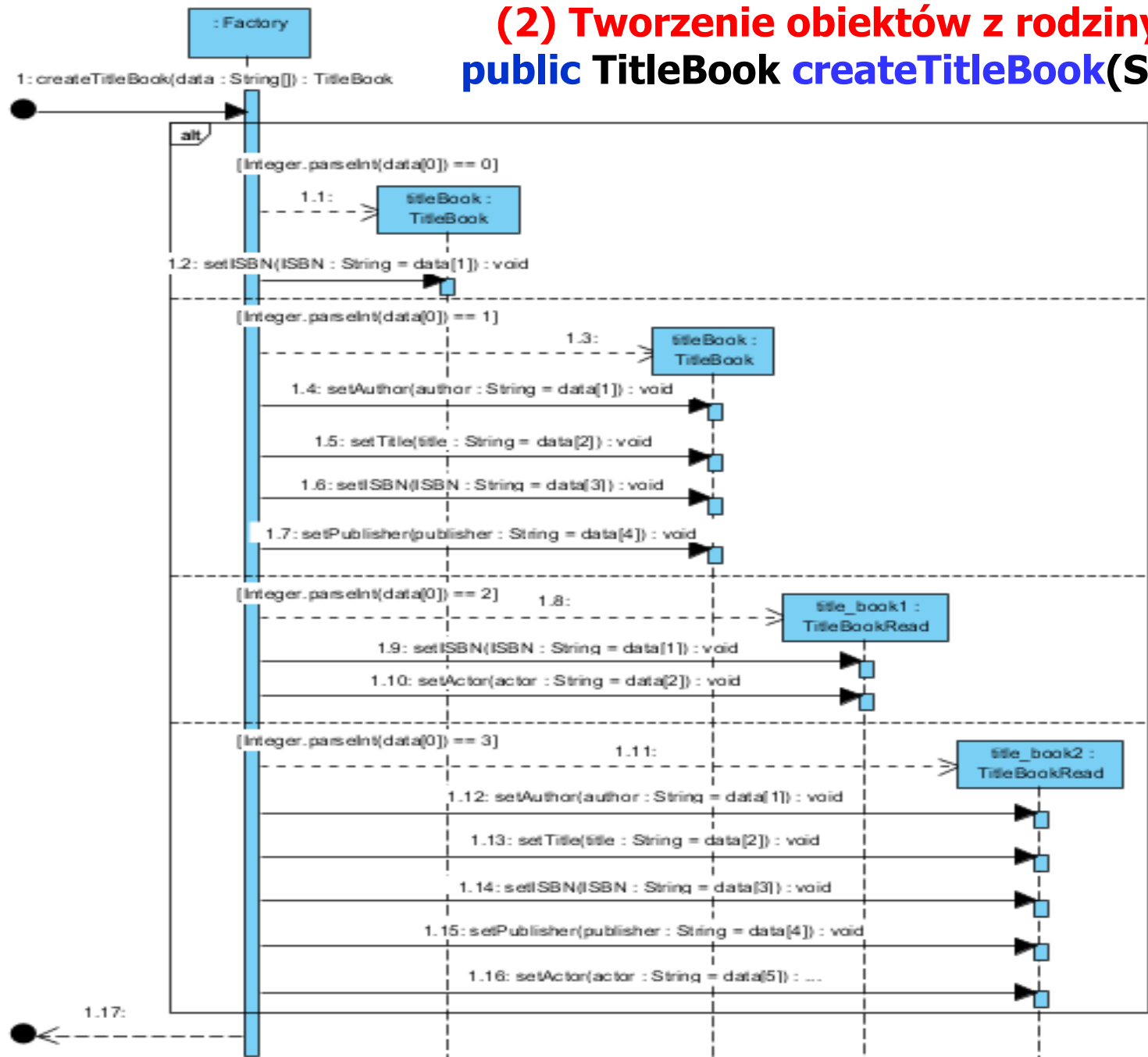
(1) Wstawianie nowego tytułu: `public String addTitleBook(String data[])`

`sd subbusinessstier.Facade.addTitleBook(String)`



(2) Tworzenie obiektów z rodziny TitleBook

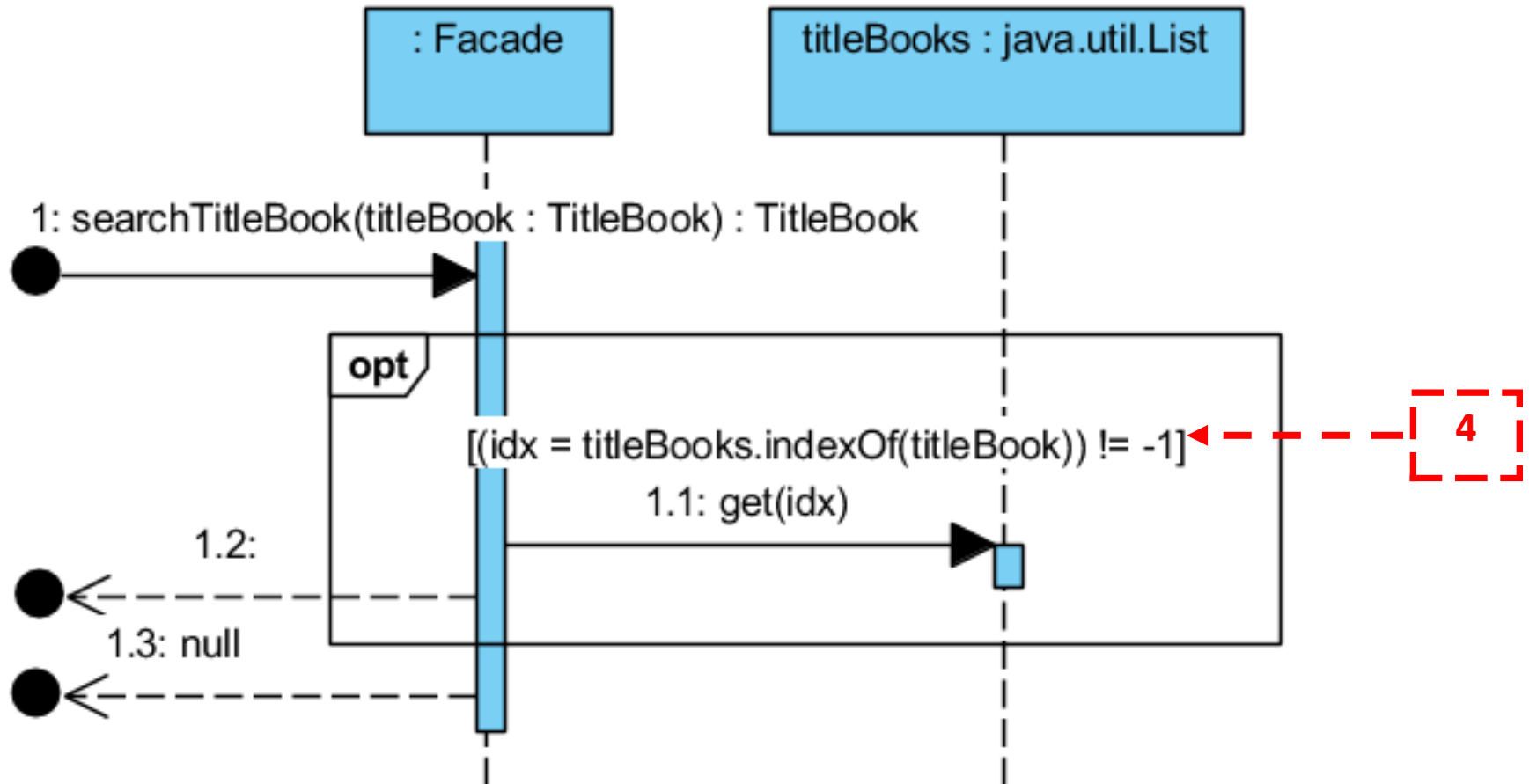
public TitleBook createTitleBook(String data[])



(3) Wyszukiwanie obiektu z rodziny TitleBook

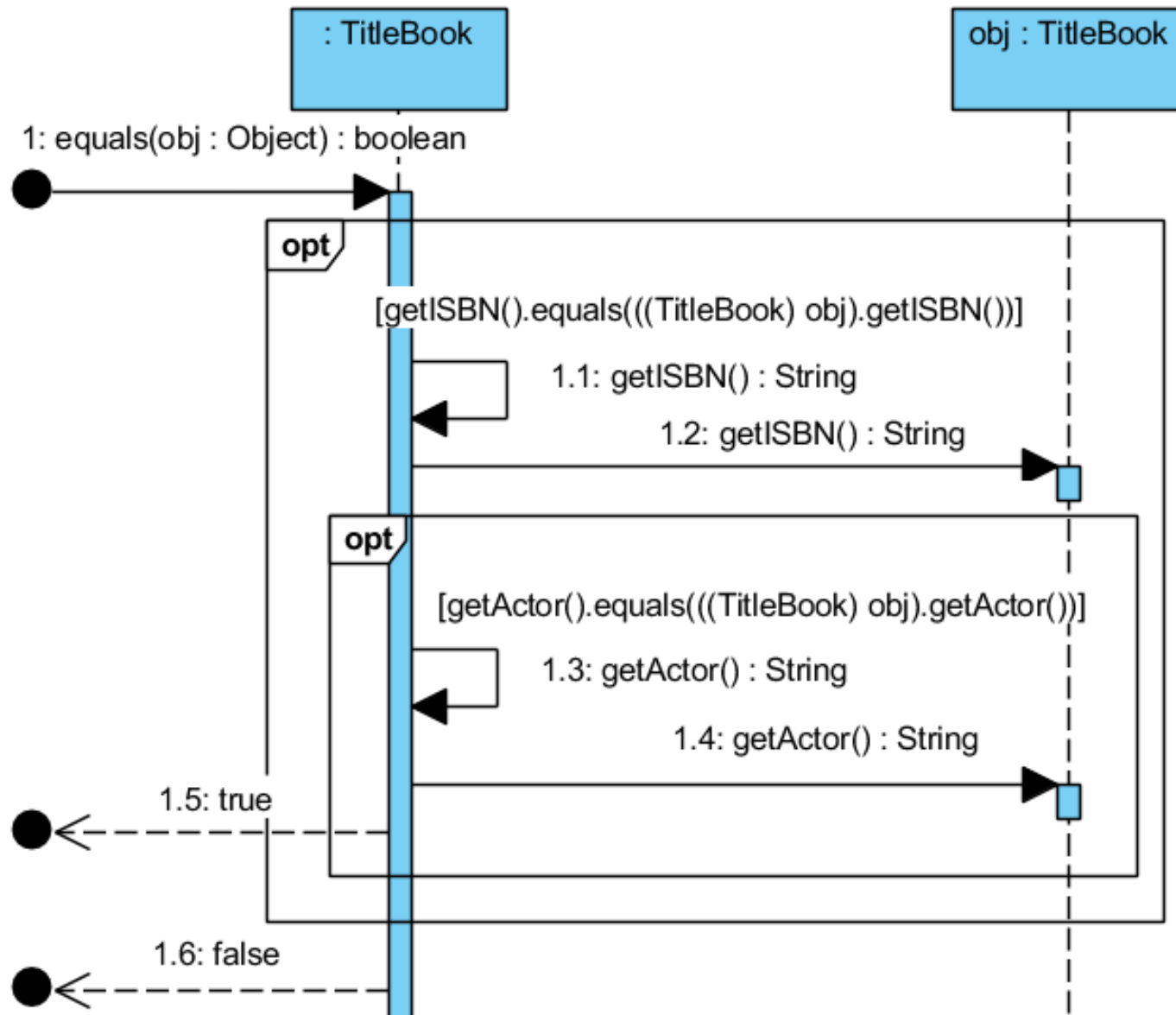
public TitleBook searchTitleBook(TitleBook titleBook)

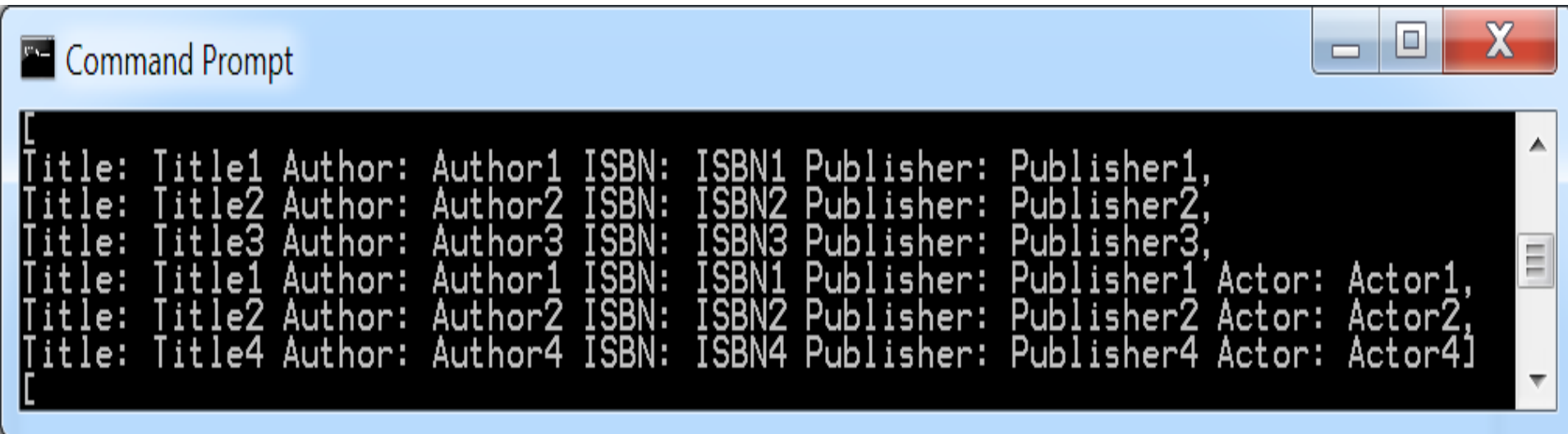
sd subbusinessstier.Facade.searchTitleBook(TitleBook)



(4) Metoda equals w klasie TitleBook: public boolean equals(Object obj)

sd subbusinessier.entities.TitleBook.equals(Object)





```
[
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1,
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2,
Title: Title3 Author: Author3 ISBN: ISBN3 Publisher: Publisher3,
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Actor: Actor1,
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2 Actor: Actor2,
Title: Title4 Author: Author4 ISBN: ISBN4 Publisher: Publisher4 Actor: Actor4]
[
```

Iteracja 2

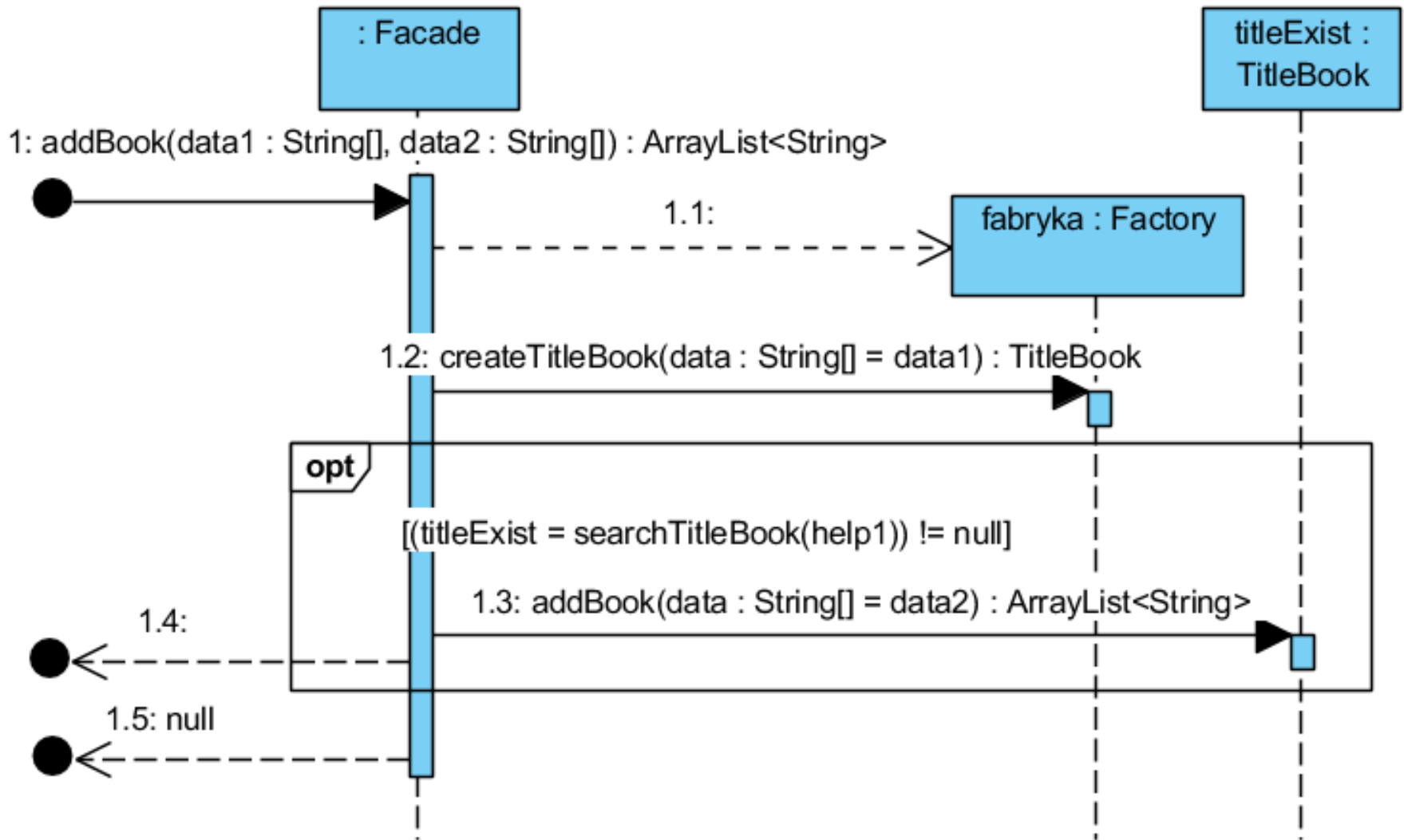
Projekt przypadku użycia

„Dodaj_Ksiazke”

za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

(5) Wstawianie nowej książki o podanym tytule – 1-y etap
`public ArrayList<String> addBook(String data1[], String data2[])`

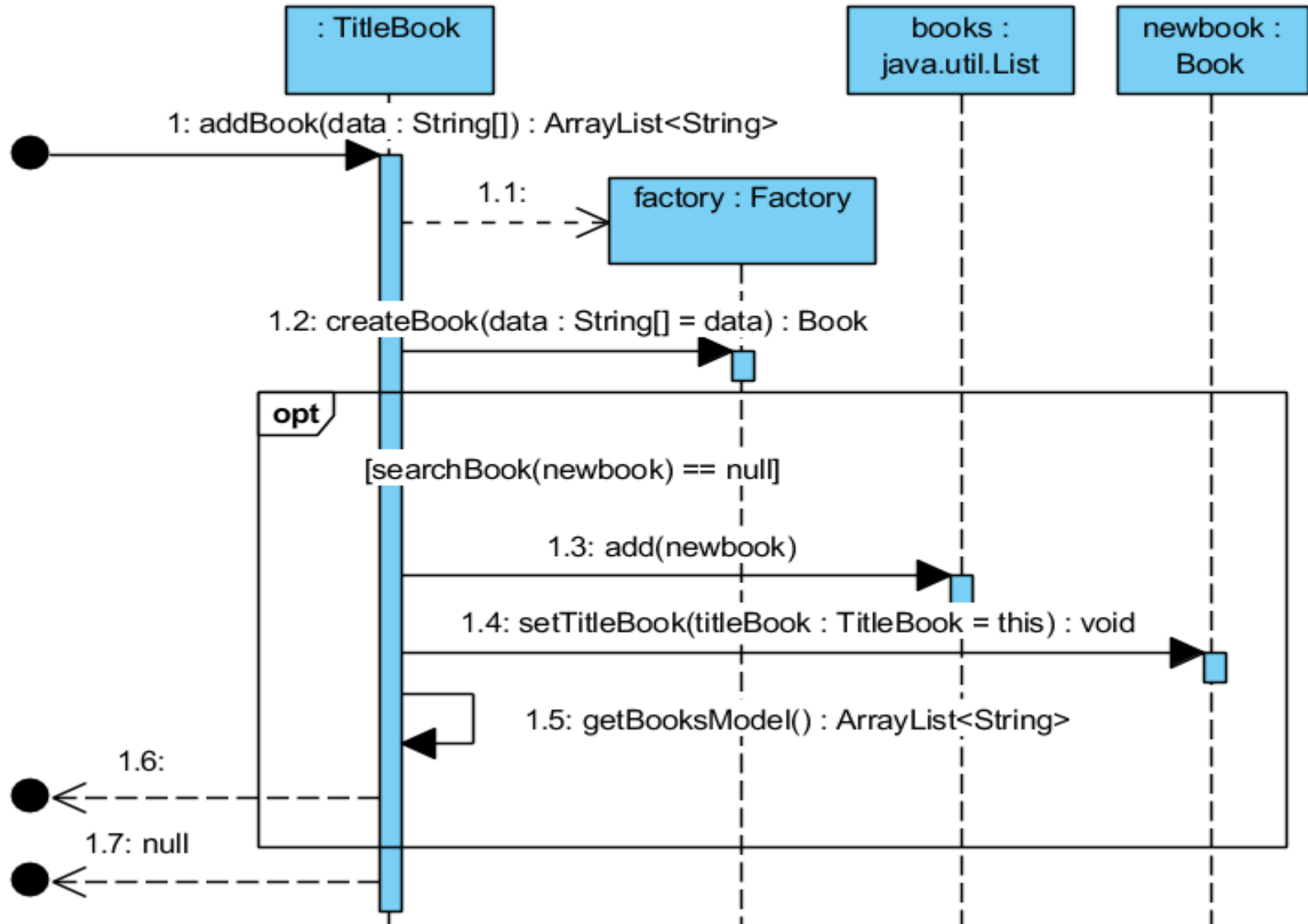
sd subbusinesssier.Facade.addBook(String, String)



(6) Wstawianie nowej książki o podanym tytule – 2-i etap

```
public ArrayList<String> addBook(String data[])
```

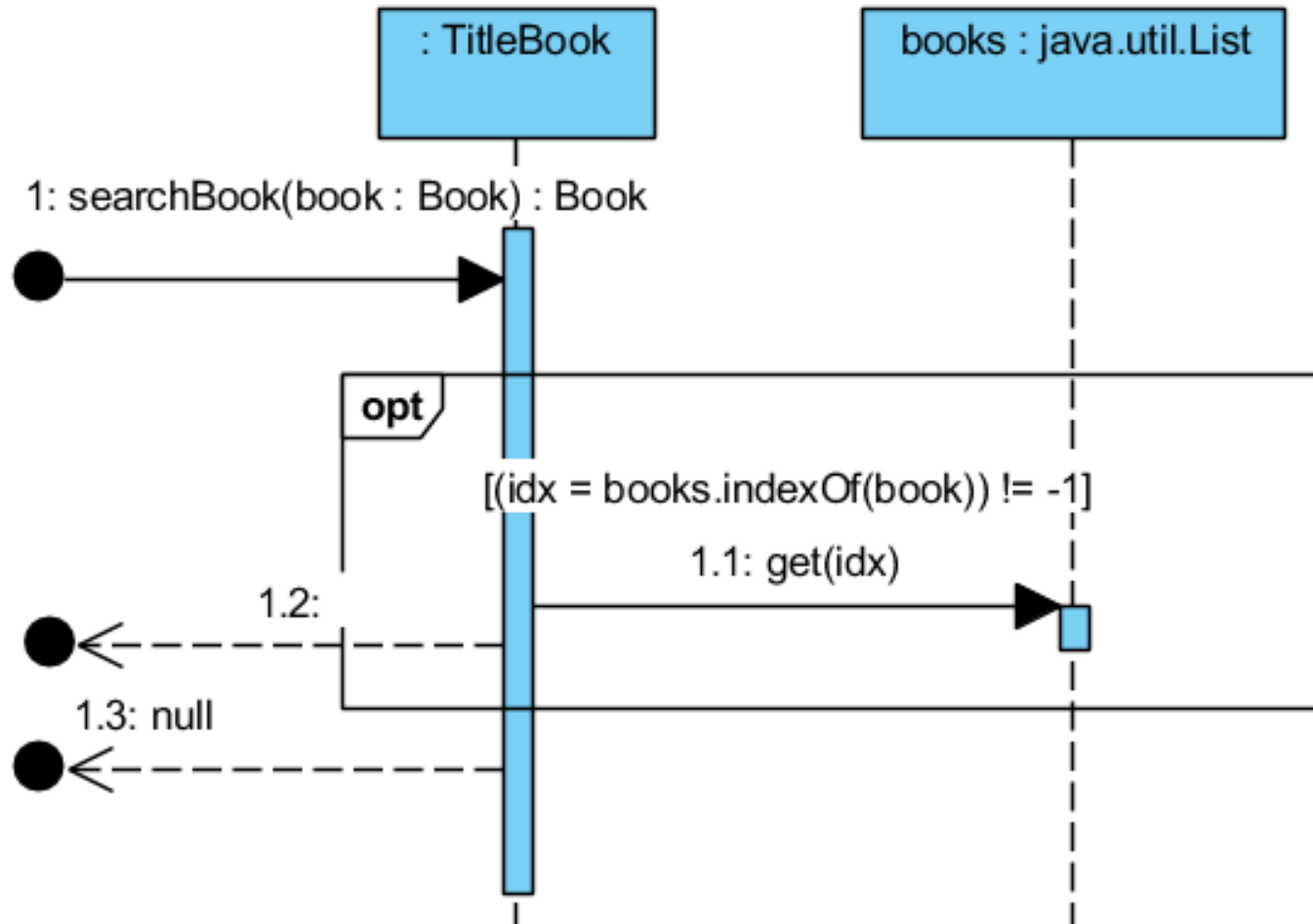
sd subbusinessstier.entities.TitleBook.addBook(String) /



(7) Szukanie książki

`public Book searchBook(Book book)`

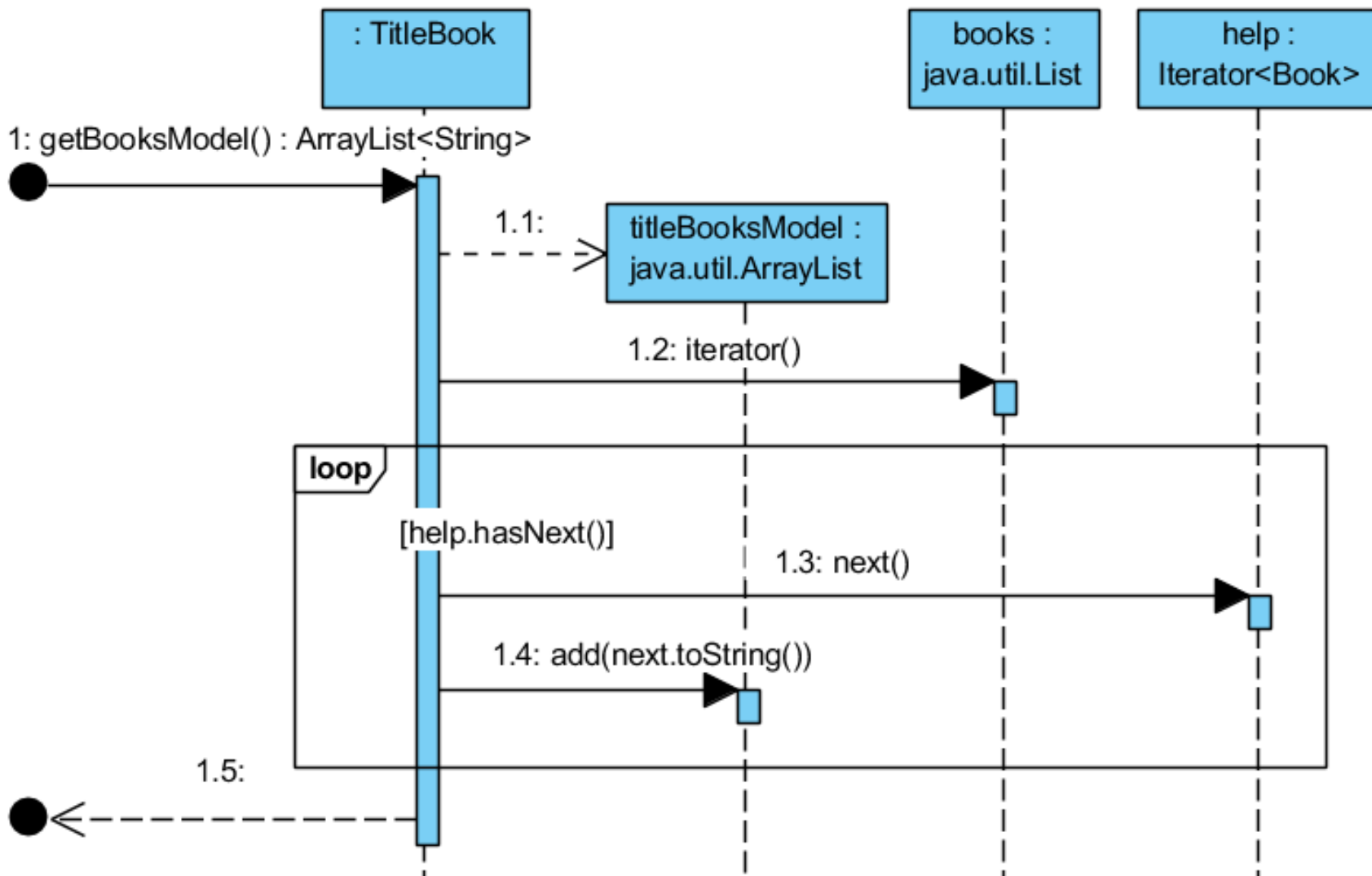
sd subbusiness-tier.entities.TitleBook.searchBook(Book)



(8) Pobranie danych o książkach należących do obiektu z rodziny TitleBook

`public ArrayList<String> getBooksModel()`

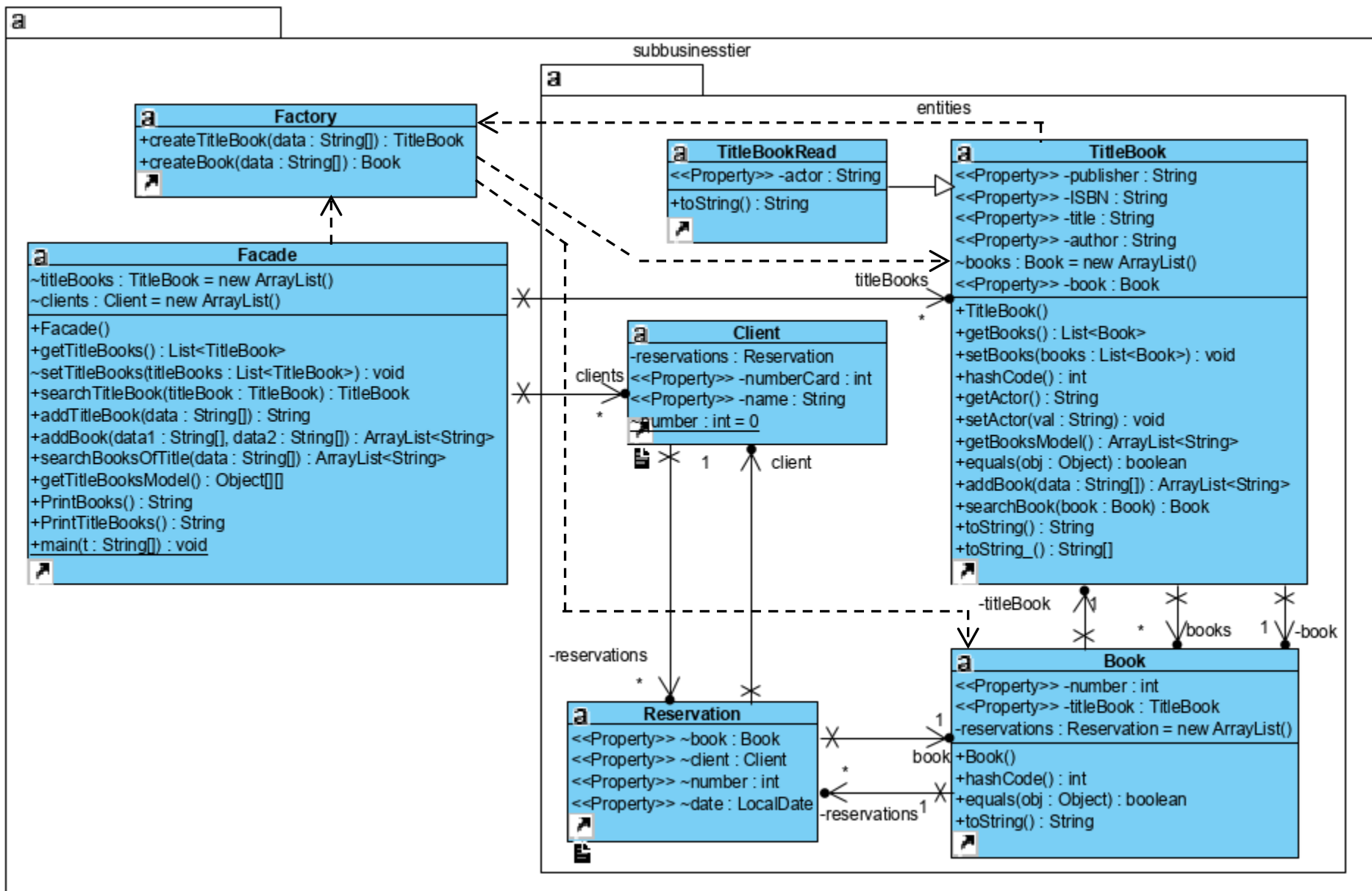
`sd subbusinessstier.entities.TitleBook.getBooksModel()`



Command Prompt

```
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1,  
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2,  
Title: Title3 Author: Author3 ISBN: ISBN3 Publisher: Publisher3,  
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Actor: Actor1,  
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2 Actor: Actor2,  
Title: Title4 Author: Author4 ISBN: ISBN4 Publisher: Publisher4 Actor: Actor4]  
[  
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Number: 1][  
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2 Number: 1][  
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2 Number: 1,  
Title: Title2 Author: Author2 ISBN: ISBN2 Publisher: Publisher2 Number: 2][  
Title: Title1 Author: Author1 ISBN: ISBN1 Publisher: Publisher1 Actor: Actor1 Number: 1][  
Title: Title4 Author: Author4 ISBN: ISBN4 Publisher: Publisher4 Actor: Actor4 Number: 2]
```

Rezultat – diagram klas uzyskany w procesie projektowania po dwóch iteracjach.



Wykonanie aplikacji dwuwarstwowej

Product Version: Apache NetBeans IDE 19
Java: 17.0.1; Java HotSpot(TM) 64-Bit Server VM 17.0.1+12-LTS-39
Runtime: Java(TM) SE Runtime Environment 17.0.1+12-LTS-39
System: Windows 10 version 10.0 running on amd64; Cp1250; pl_PL (nb)
User directory: C:\Users\Zofia\AppData\Roaming\NetBeans\19
Cache directory: C:\Users\Zofia\AppData\Local\NetBeans\Cache\19

Warstwa klienta
-aplikacja desktopowa (**GUI**)
do wykonania
Library1_client1_SE



Warstwa biznesowa –
przetwarzanie danych.
Projekt typu
Java Class Library,
używany jako
biblioteka logiki
biznesowej, który
zawiera kod utworzony
podczas 2 iteracji-
Library_1IO

Uwaga: z klasy **Facade** opartej na wzorcu **Fasada**, usunięto metodę **main**, która służyła jedynie do ręcznego testowania projektowanego kodu **Warstwy biznesowej**

Projekt typu
Java Class Library

zawierający kod
Warstwy biznesowej
wykonany podczas
dwóch iteracji

Projekt typu **Java
Application**

zawierający kod
Warstwy klienta
desktopowego z
interfejsem
graficznym
użytkownika (**GUI**)
do kodu 1-ej i 2-ej
iteracji

The screenshot shows an IDE with two projects:

- Library_1IO**: A Java Class Library project containing source packages (subbusinessstier, subbusinessstier.entities) and test packages. Source files include Facade.java, Factory.java, Book.java, Client.java, Reservation.java, TitleBook.java, and TitleBookRead.java.
- Library1_client1_SE**: A Java Application project containing source packages (client_tier, library) and test packages. Source files include Client.java, Book_form.java, Card0.java, Loan_form.java, Panel_util.java, and Title_form.java. A red box highlights the file `Library_1IO - dist/Library_1IO.jar` in the Libraries section.

A red box highlights the `Facade.java` file in the `subbusinessstier` package. A text box next to it states: "Projekt zawierający *Warstwę biznesową* jest biblioteką dla projektu *Warstwy klienta*". A red arrow points from the GUI jar file in the second project to this text box.

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12 List<TitleBook> titleBo  
13 List<Client> clients;  
14  
15 public Facade(A...
```

run:
BUILD SUCCESSFUL (total time: 8 second

Projects × Files Services

- Library_110
 - Source Packages
 - subbusinesssier
 - Facade.java
 - Factory.java
 - subbusinesssier.entities
 - Book.java
 - Client.java
 - Reservation.java
 - TitleBook.java
 - TitleBookRead.java
 - Test Packages
 - Libraries
 - JDK 1.8
 - Test Libraries
 - Library1_client1_SE
 - Source Packages
 - client_tier
 - Client.java
 - library
 - Book_form.java
 - Card0.java
 - Loan_form.java
 - Panel_util.java
 - Title_form.java
 - Test Packages
 - Libraries
 - Library_110 - dist/Library_110.jar
 - JDK 1.8
 - Test Libraries

```
Facade.java × Client.java ×
Source History
1 package client_tier;
2
3 import library.Panel_util;
4 import subbusinesssier.Facade;
5
6
7 public class Client {
8
9     static Facade facade = new Facade();
10
11     static public Facade getFacade() {
12         return facade; }
13
14     static public void setFacade(Facade facade)
15         Client.facade = facade; }
16
17     public static void main(String[] args) {
18         java.awt.EventQueue.invokeLater(() -> {
19             Panel_util.createAndShowGUI();
20         });
21     }
22 }
23
```

Udostępnianie w programie *Warstwy klienta* logiki biznesowej za pomocą wzorca **Facade, występującego w roli wzorca **Singleton****

static Facade facade = new Facade();

static public Facade getFacade() {
return facade; }

static public void setFacade(Facade facade)
Client.facade = facade; }

Udostępnianie logiki biznesowej w **Warstwie klienta** (formularz do wprowadzania danych tytułu) za pomocą metody **addTitleBook** wzorca **Facade**.

```
50
51 @Override
52 public void actionPerformed(ActionEvent evt) {
53     String[] data = form_title();
54     if (data == null) {
55         return;
56     }
57     Client.getFacade().addTitleBook(data);
58 }
59
60 public String[] form_title() {
61     if (content_validate(val:title) == null) {
62         return null;
63     }
64     if (content_validate(val:ISBN) == null) {
65         return null;
66     }
67     if (content_validate(val:publisher) == null) {
68         return null;
69     }
70     if (content_validate(val:author) == null) {
71         return null;
72     }
73 }
```

Search Results

Output x

>>

< abase Process x

GlassFish Server5 x

Library1_client1_SE (run) x



100:1

INS



Wynik metody **getTitleBookModel** jest wykorzystany jako **model widoku JTable**

```
void table content() {
    Object[][] titles = Client.getFacade().getTitleBooksModel();
    model.setData(titles);
}

@Override
public void actionPerformed(ActionEvent event) {
    if (event.getSource() == clear_selection) {
        table.getSelectionModel().clearSelection();
        return;
    }
    if (!table.getSelectionModel().isSelectionEmpty()) {
        String what_book_type;
        if (number.getText().equals("")) {
            JOptionPane.showMessageDialog(this, "required value");
            return;
        }
        what_book_type = "0";
        String data2[] = {what_book_type, (String) number.getText()};
        ArrayList<String> help3 = Client.getFacade().addBook(title(), data2);
        if (help3 != null) {
            list_content(help3, books);
        }
        table.getSelectionModel().clearSelection();
    }
}
```

Udostępnianie logiki biznesowej w **Warstwie klienta** (formularz do wprowadzania danych książki) za pomocą metody **addBook** wzorca **Facade**,

`ArrayList<String> help3 = Client.getFacade().addBook(title(), data2);`

`list_content(help3, books);`

Wynik **help3** metody **addBook** jest wykorzystany jako **model widoku JComboBox**

```
private void list_content(ArrayList<String> col, JComboBox list)
{
    String s;
    list.removeAllItems();
    Iterator<String> iterator = col.iterator();
    while (iterator.hasNext()) {
        s = iterator.next();
        list.addItem(s);
    }
}
```

Formularze do wprowadzanie danych tytułów (z lewej) i książek (z prawej)

MenuDemo

A Menu Another Menu

Title

Author

ISBN

Publisher

Actor

Add title

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytuł1	Autor1	Aktor1

Number of a book

Add book

Clear selection

Books

MenuDemo

A Menu Another Menu

Title

Author

ISBN

Publisher

Actor

Add title

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytuł1	Autor1	Aktor1
Wydawca1	1234567	Tytuł1	Autor1	

Number of a book

Add book

Clear selection

Books

Wprowadzanie danych książek papierowych i nagranych (cd)

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytul1	Autor1	Aktor1
Wydawca1	1234567	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

▼

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytul1	Autor1	Aktor1
Wydawca1	1234567	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

Title: Tytul1 Author: Autor1 ISBN: 1234567 Publisher: Wydawca1 Actor: Aktor1 Number: 1 ▼

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytul1	Autor1	Aktor1
Wydawca1	1234567	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

▼

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	1234567	Tytul1	Autor1	Aktor1
Wydawca1	1234567	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

Title: Tytul1 Author: Autor1 ISBN: 1234567 Publisher: Wydawca1 Number: 2 ▼

Title: Tytul1 Author: Autor1 ISBN: 1234567 Publisher: Wydawca1 Number: 2

Zagadnienia

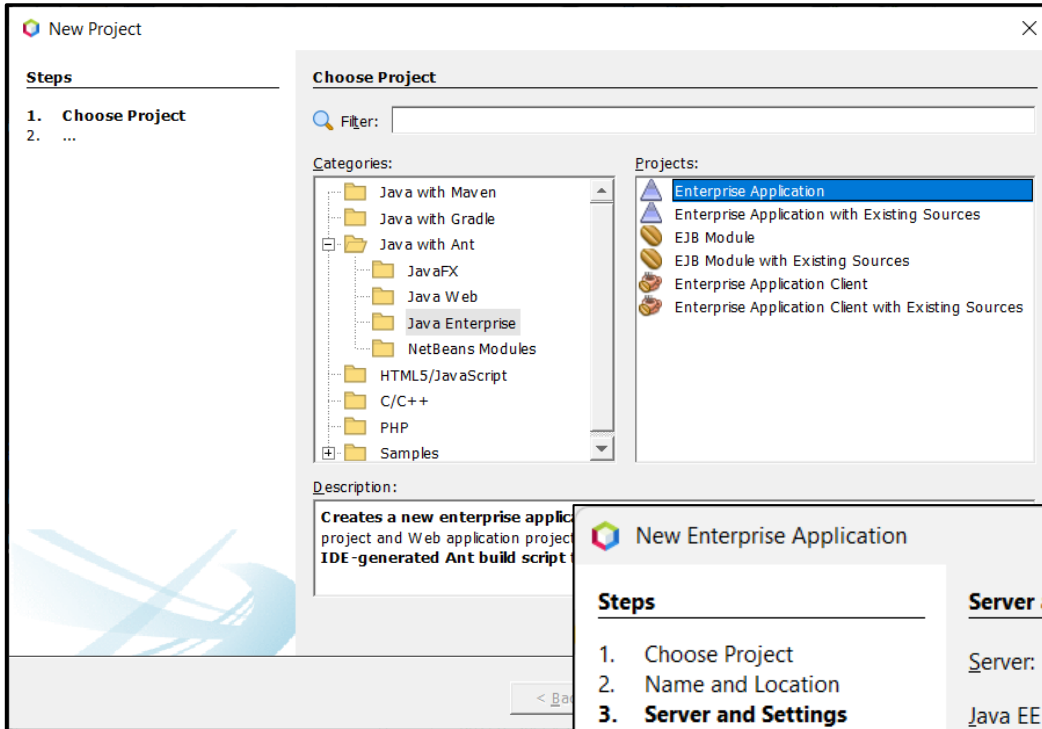
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
6. **Przykłady architektury wielowarstwowej aplikacji typu EE (p.2). Wykonanie aplikacji typu EE. Warstwa biznesowa: komponenty typu EJB + obiekty POJO**

Architektura oprogramowania

Pięciowarstwowy model logicznego rozdzielania zadań aplikacji
(wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

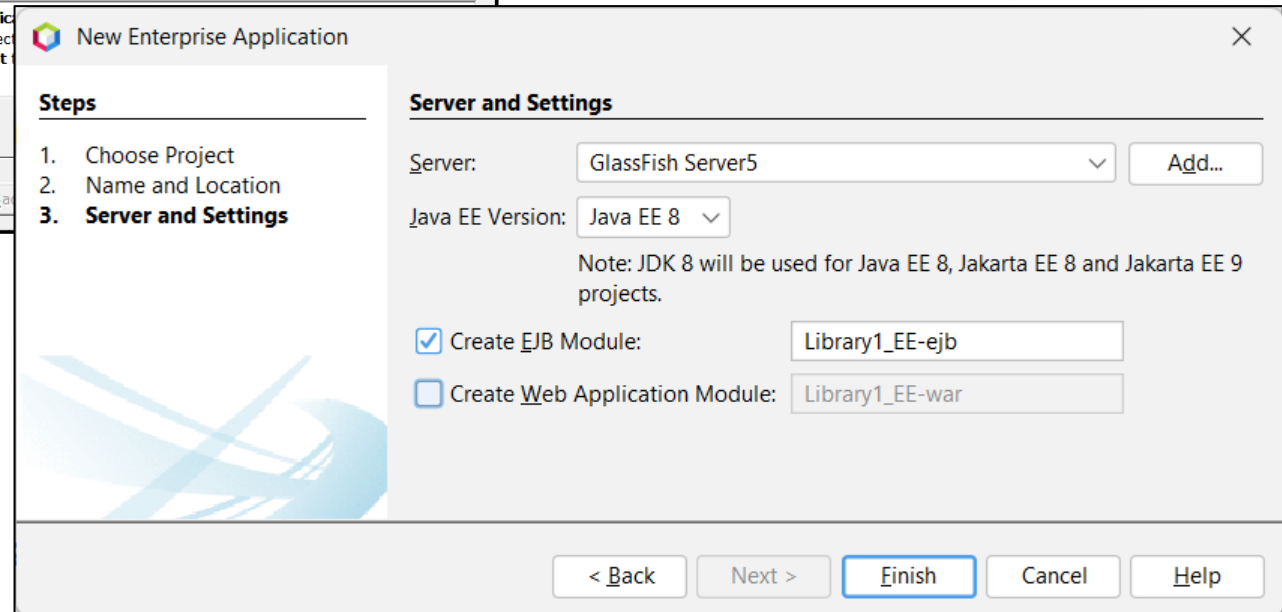


Wykonanie aplikacji typu **Java EE** z modułem **EJB** typu **Stateless** (slajd 14).
W tym module należy umieścić komponent typu **EJB** w celu umożliwienia
zdalnego dostępu do metod obiektu typu **Facade** przez wiele aplikacji
reprezentujących **Warstwę klienta**: desktopowych i internetowych

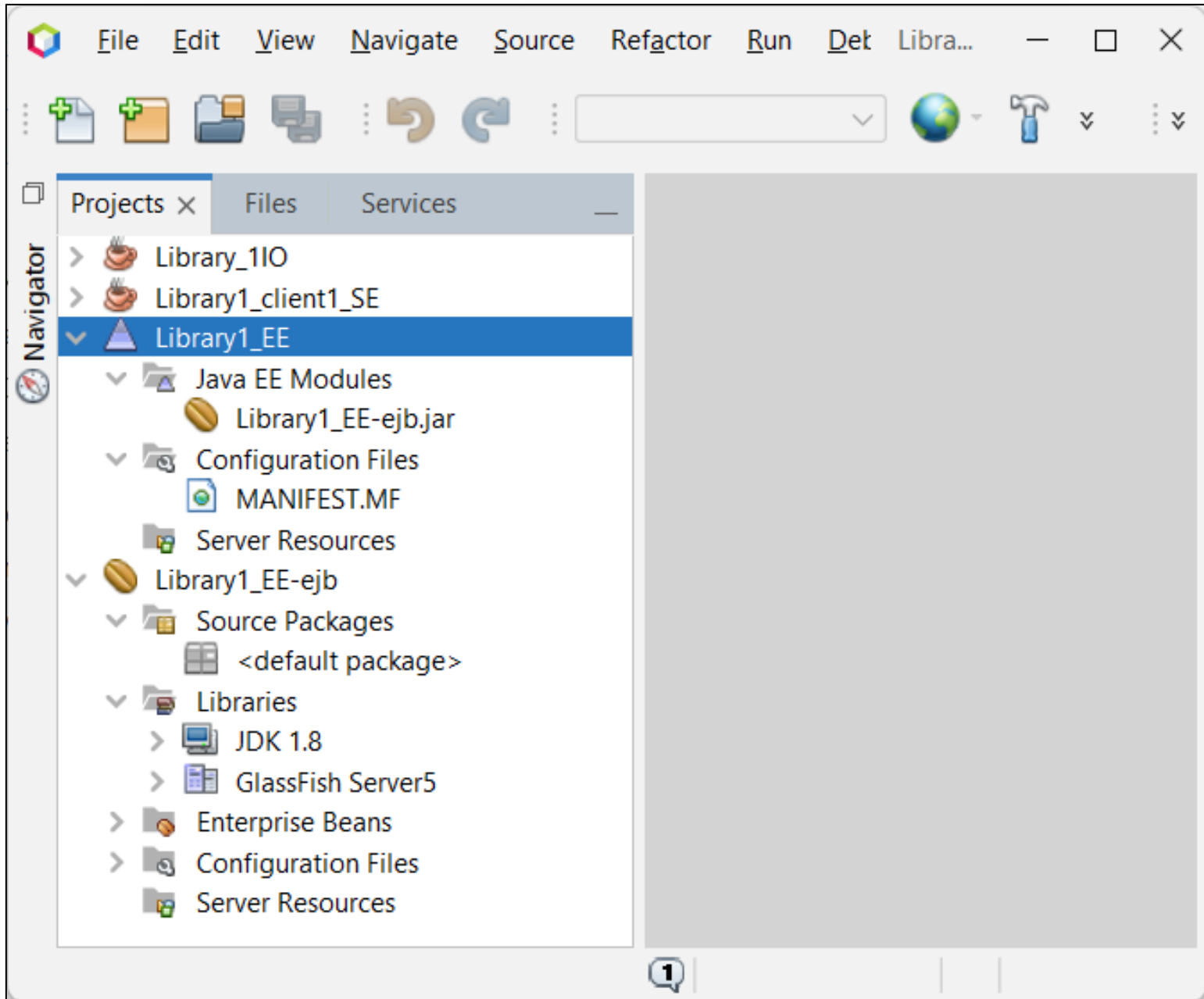


1. Tworzenie aplikacji typu
Enterprise Application:
Library1_EE

2. Dodanie modułu **EJB**
w aplikacji
Enterprise Application:
Library1_EE-ejb



3. Projekt typu **Enterprise Application**



4. Wstawianie zdalnego komponentu EJB typu Session Bean do modułu EJB

The image displays four sequential screenshots from the NetBeans IDE, illustrating the process of creating a new project and then a new session bean.

Top Left: New Project dialog
The "Choose Project" step shows the "Java Class Library" option selected in the "Projects" list. The description states: "Creates a new Java SE library in a standard IDE project. A Java SE library does not contain a main class. Standard projects use an IDE-generated Ant build script to build, run, and debug your project."

Top Right: Library Location dialog
The "Library1_EE_Interface" dialog shows the file path: "C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\programy\library1\Library1_EE_Interface".

Bottom Left: New File dialog
The "Choose File Type" step shows "Session Bean" selected in the "File Types" list. The description states: "Creates an empty Session Enterprise JavaBean (EJB) component. A session bean is typically used to encapsulate business logic or enterprise resources. This template creates the Java classes for a single session bean and registers the bean in the EJB".

Bottom Right: New Session Bean dialog
The "Name and Location" step shows the "EJB Name" as "EJBFacade", the "Project" as "Library1_EE-ejb", and the "Package" as "busnesstier". The "Session Type" is set to "Stateless". The "Create Interface" section has "Remote in project" checked, with "Library1_EE_Interface" selected in the dropdown.

5. Wygenerowany: zdalny komponent **Session Bean: EJBFacade** w **Library1_EE-ejb** i jego interfejs **EJBFacadeRemote** w projekcie **Library1_EE_Interface**

The screenshot shows the Eclipse IDE interface. On the left, the Navigator view displays the project structure for 'Library1_EE'. The 'businessstier' package is expanded, showing the 'EJBFacadeRemote.java' file. The main editor window displays the source code of 'EJBFacadeRemote.java' with the following content:

```
1  ...4 lines
5  package businessstier;
6
7  import javax.ejb.Remote;
8
9  /**...4 lines */
13 @Remote
14 public interface EJBFacadeRemote {
15
16 }
17
```

The status bar at the bottom indicates the cursor is at line 16, column 2, in the 'businessstier.EJBFacadeRemote' file.

This screenshot shows the Eclipse IDE interface with the 'EJBFacade.java' file open in the editor. The code is as follows:

```
...4 lines
package businessstier;

import javax.ejb.Stateless;

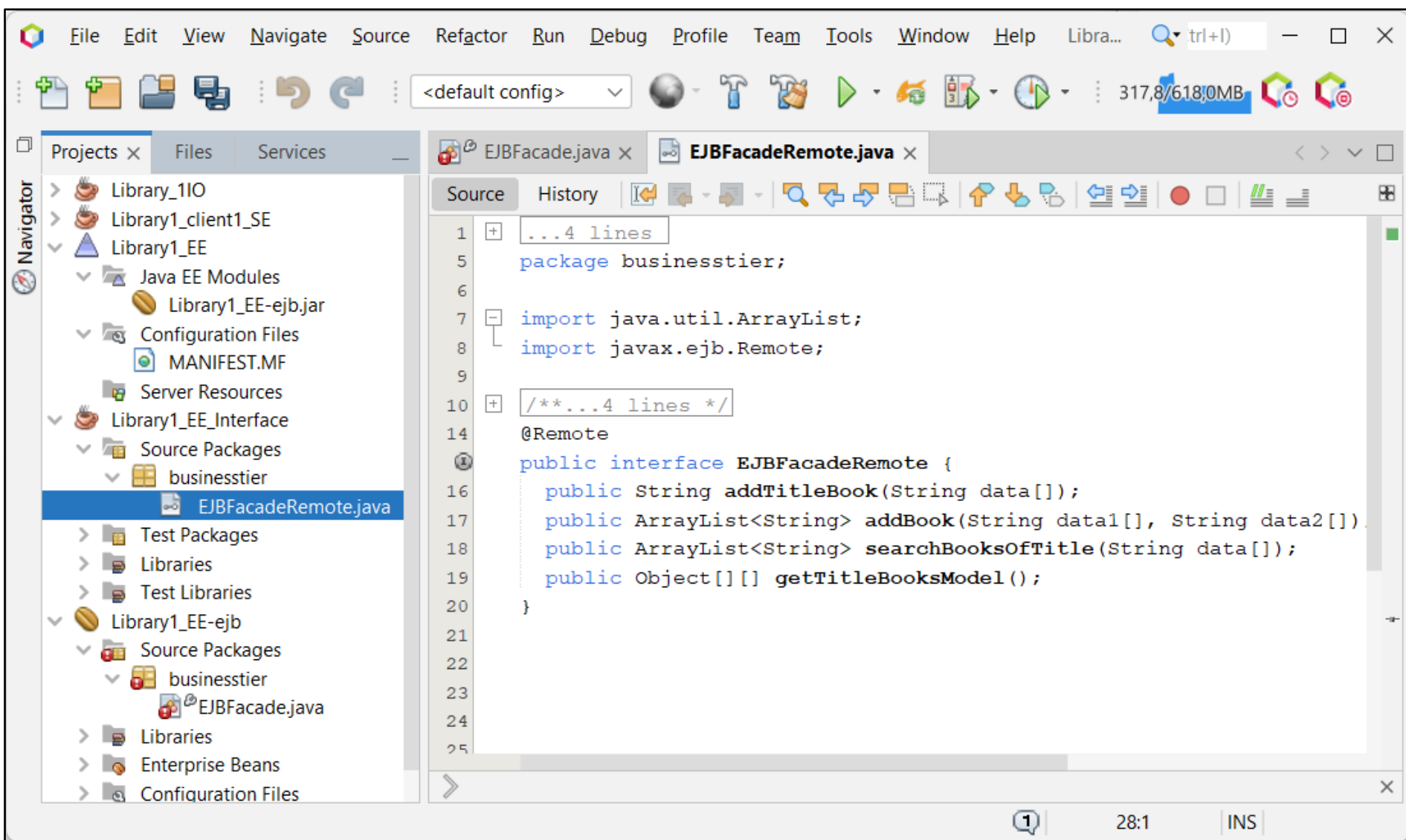
/**...4 lines */
@Stateless
public class EJBFacade implements EJBFacadeRemote {

    // Add business logic below. (Right-click in editor
    // "Insert Code > Add Business Method")

}
```

The status bar at the bottom indicates the cursor is at line 19, column 1, in the 'EJBFacade' class.

6. Deklaracja metod o zdalnym dostępie do metod logiki biznesowej klasy **Facade** w interfejsie komponentu typu **Session Bean**



The screenshot displays an IDE window with the following components:

- Navigator:** Shows a project structure with folders like Library_11IO, Library1_client1_SE, Library1_EE, and sub-packages such as Java EE Modules, Configuration Files, Server Resources, Source Packages (including `businessstier`), and Test Packages.
- Editor:** Displays the `EJBFacadeRemote.java` file with the following code:

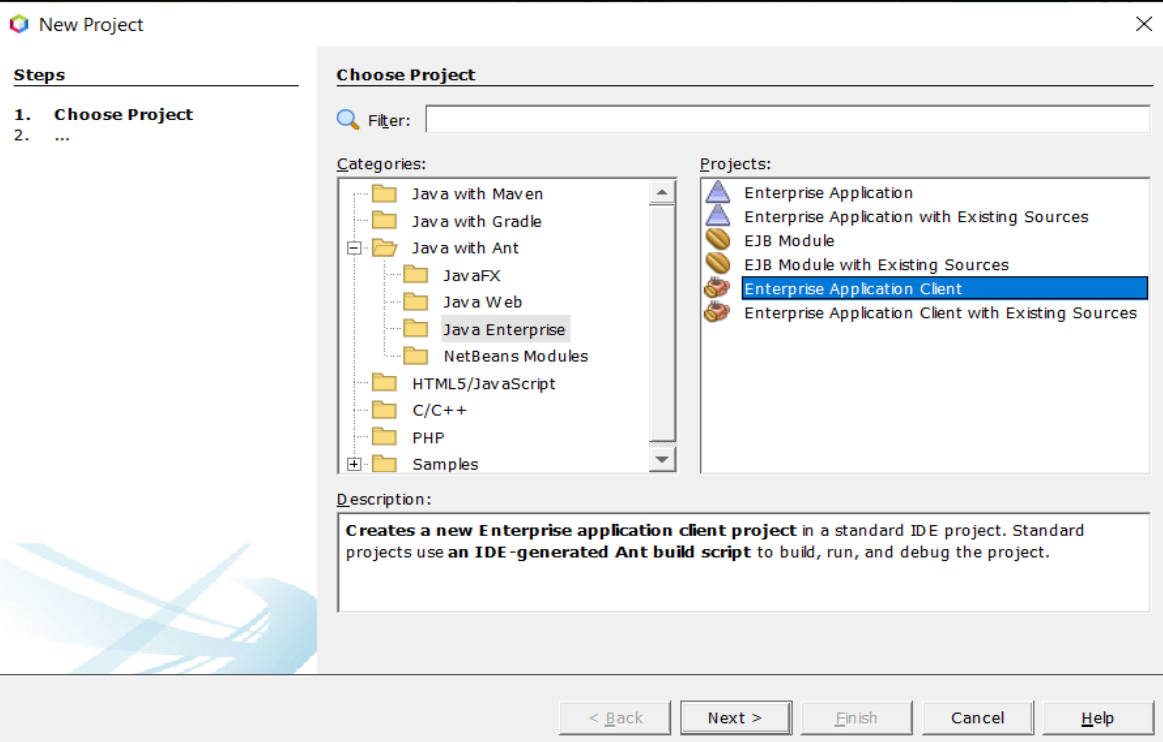
```
1  ...4 lines
5  package businessstier;
6
7  import java.util.ArrayList;
8  import javax.ejb.Remote;
9
10 /** ...4 lines */
14 @Remote
16 public interface EJBFacadeRemote {
17     public String addTitleBook(String data[]);
18     public ArrayList<String> addBook(String data1[], String data2[]);
19     public ArrayList<String> searchBooksOfTitle(String data[]);
20     public Object[][] getTitleBooksModel();
21 }
22
23
24
25
```
- Toolbar:** Includes standard IDE icons for file operations, navigation, and execution, along with a configuration dropdown set to `<default config>` and a memory usage indicator showing `317,8/618,0MB`.

7. Definicja metod o zdalnym dostępie do metod logiki biznesowej klasy Facade (wzorzec **ApplicationService**) w komponencie **Session Bean** (wzorzec **SessionFacade**)

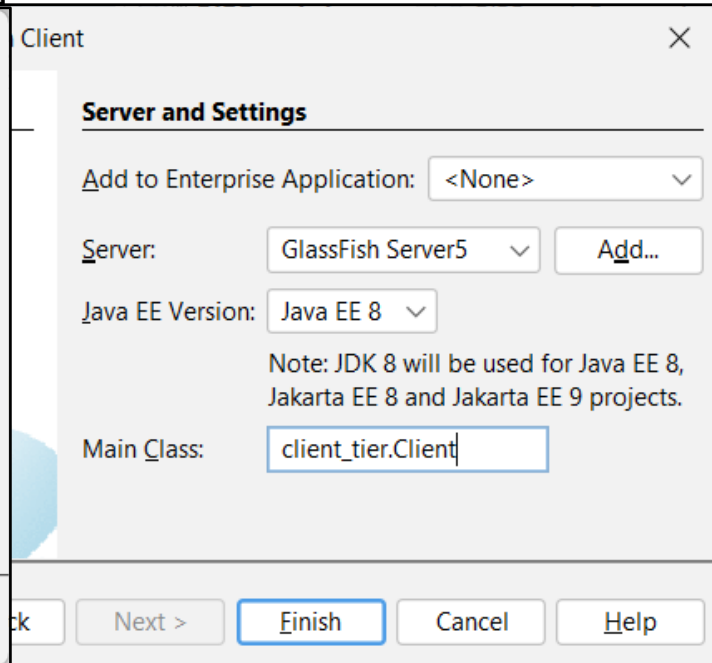
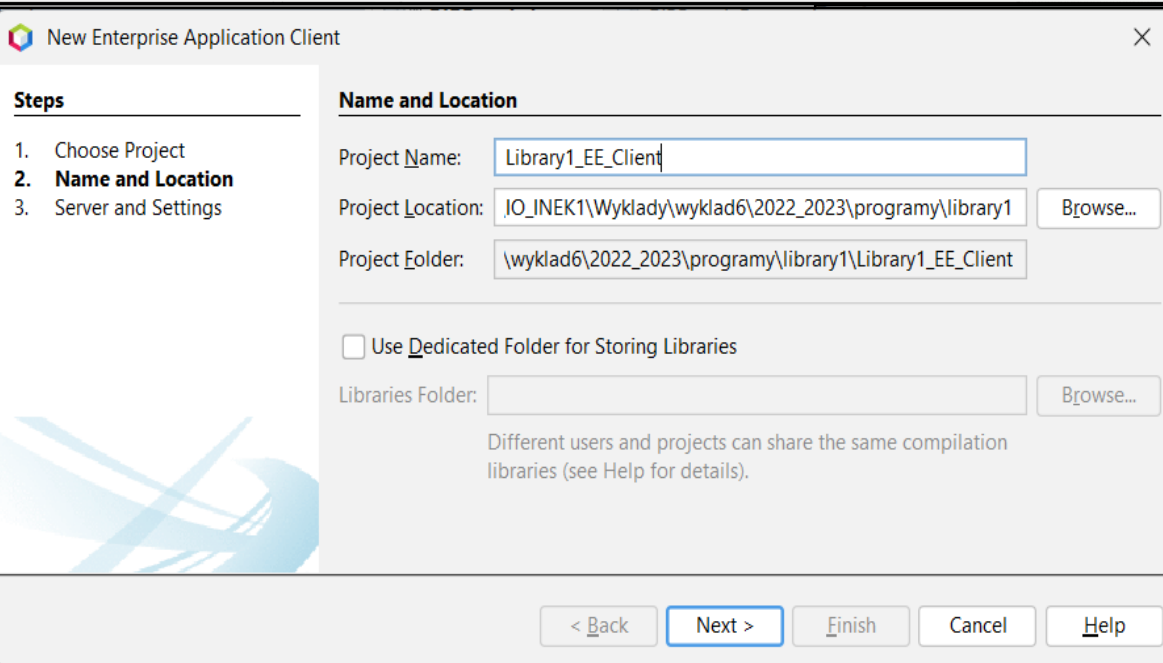
The screenshot shows an IDE window with the following components:

- Navigator:** Shows a project structure with packages like `businessstier` and `EJBFacadeRemote.java`.
- Source Editor:** Displays the code for `EJBFacade.java`. The code includes imports for `ArrayList`, `Stateless`, and `Facade`. It defines a `@Stateless` class `EJBFacade` that implements `EJBFacadeRemote`. A red box highlights the line `Facade facade = new Facade();` at line 17. A red arrow points from a callout box to this line.
- Callout Box:** A red-bordered box containing the text: **Obiekt typu *Facade* pełni rolę wzorca *ApplicationService***.

```
5 package businessstier;
6
7 import java.util.ArrayList;
8 import javax.ejb.Stateless;
9 import subbusinessstier.Facade;
10
11 /**...4 lines */
12
13 @Stateless
14
15 public class EJBFacade implements EJBFacadeRemote {
16     Facade facade = new Facade();
17
18     @Override
19     public String addTitleBook(String data[]) {
20         return facade.addTitleBook(data);
21     }
22
23     @Override
24     public ArrayList<String> addBook(String data1[], String data2[]) {
25         return facade.addBook(data1, data2);
26     }
27
28     @Override
29     public ArrayList<String> searchBooksOfTitle(String data[]) {
30         return facade.searchBooksOfTitle(data);
31     }
32
33     @Override
34     public Object[][] getTitleBooksModel() {
35         return facade.getTitleBooksModel();
36     }
37 }
```

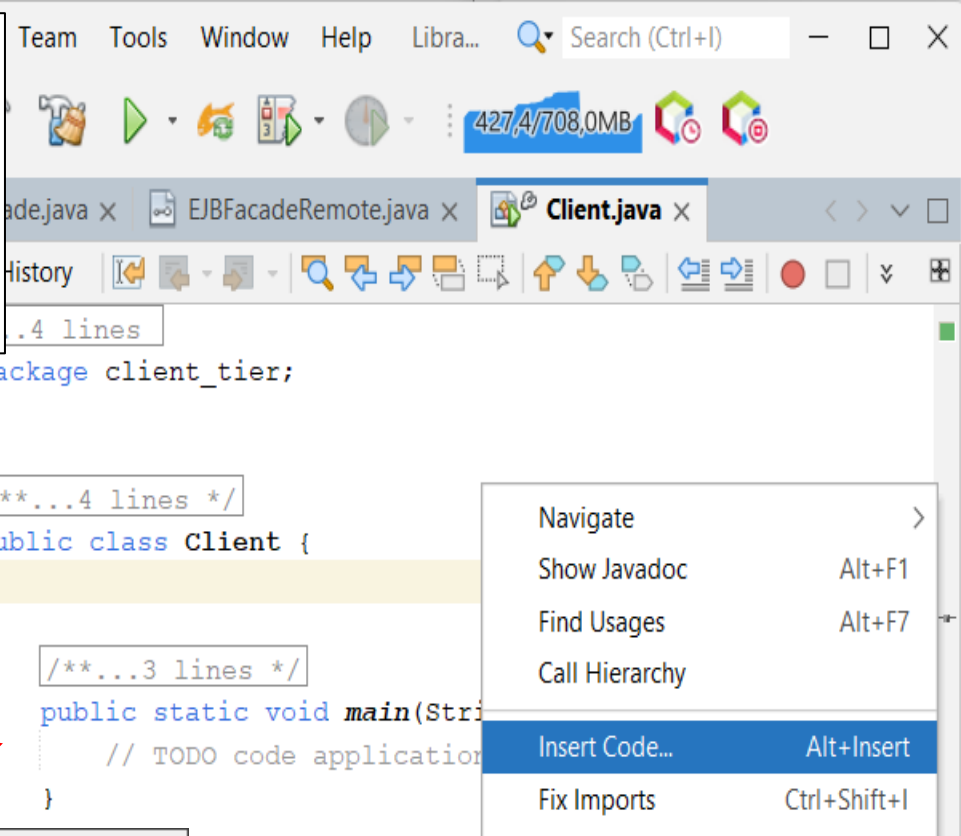
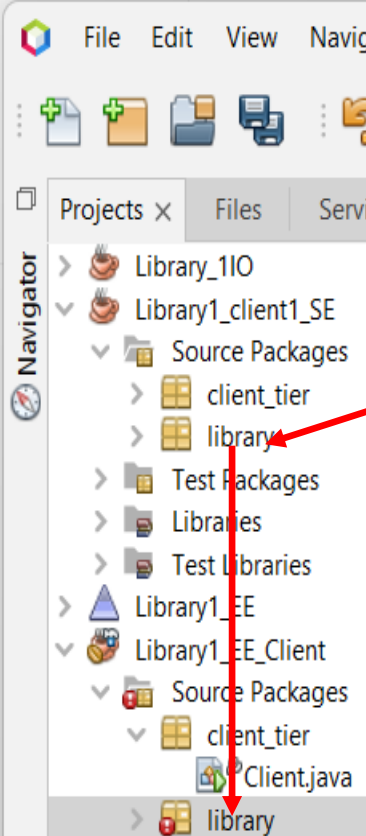


8. Wykonanie projektu typu **Enterprise Application Client**. Kod tego projektu będzie wykonany w oparciu o pakiet **library** z projektu **Warstwy klienta** (aplikacja 2-warstwowa). Zdefiniowano domyślną klasę **Client** w pakiecie **client_tier** w celu uniknięcia modyfikacji kodu klas w pakiecie **library**.

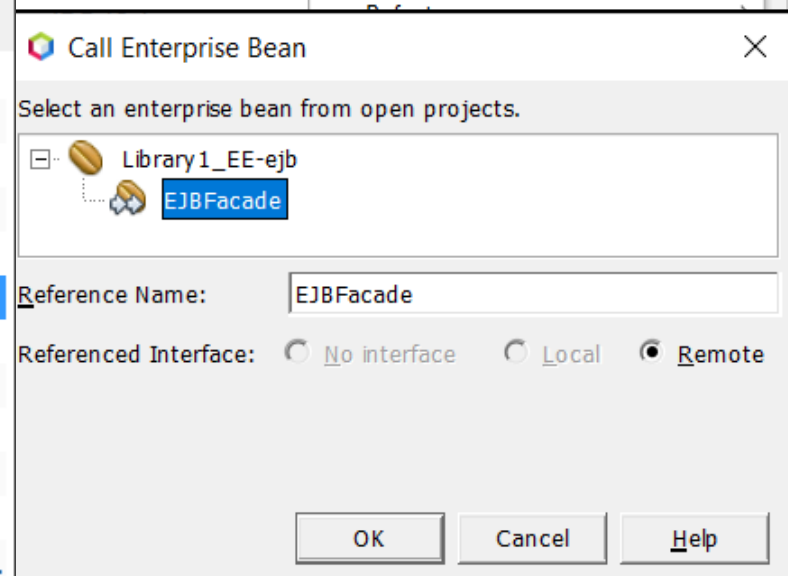
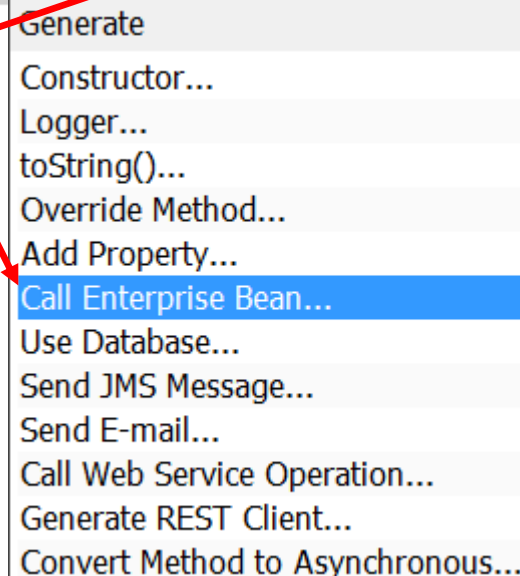


9. Skopiowanie pakietu

library z aplikacji reprezentującej aplikację **Warstwy klienta** z p.5.



10. „Wstrzyknięcie” dostępu do obiektu typu **Session Bean** w projekcie w typu **Enterprise Application Client** (w kodzie klasy **Client**)



11. Wykonana aplikacja `Library1_EE_Client` reprezentuje *Warstwę klienta EE*

Dostęp do logiki biznesowej za pomocą wzorca **SessionFacade**, występującego również w roli wzorca **Singleton (komponent EJB)**

```
7  import business.tier.EJBFacadeRemote;
8  import javax.ejb.EJB;
9  import library.Panel_util;
10
11  /**...4 lines */
15  public class Client {
16      @EJB
17      private static EJBFacadeRemote facade;
18
19      public static EJBFacadeRemote getFacade() {
20          return facade;
21      }
22
23      /**...3 lines */
26      public static void main(String[] args) {
27          // TODO code application logic here
28          java.awt.EventQueue.invokeLater(() -> {
29              Panel_util.createAndShowGUI();
30          });
31      }
32
33  }
34
35
```

12. Uruchomienie desktopowej aplikacji wielowarstwowej typu EE:
Deploy na aplikacji typu **Enterprise Application (Library1_EE)**, a potem **Run** na projekcie typu **Enterprise Application Client (Library1_EE_Client)**. Uruchomienie kolejnej aplikacji klienta wymaga jedynie powtórzenia operacji **Run**.

The screenshot displays an IDE interface with the following components:

- Navigator:** Shows a project tree with 'Library1_EE' selected. A context menu is open over it, with 'Deploy' highlighted in blue and a red rectangle around it.
- Source Editor:** Displays the code for 'Client.java'. The code includes imports for 'businessstier.EJBFacadeRemote', 'javax.ejb.EJB', and 'library.Panel_util'. It defines a 'public class Client' with an '@EJB' annotation and a 'private static EJBFacadeRemote facade;' field. A 'public static EJBFacadeRemote getFacade()' method is also shown.
- Output Console:** Shows the build process for 'Library1_EE (clean,dist)'. The output includes the following steps and messages:

```
Building jar: C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\prc
Library1_EE-ejb.dist-ear:
pre-pre-compile:
pre-compile:
Copying 1 file to C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023
do-compile:
Copying 1 file to C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023
post-compile:
compile:
pre-dist:
do-dist-without-manifest:
do-dist-with-manifest:
Created dir: C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\prog
Building jar: C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\prc
post-dist:
dist:
BUILD SUCCESSFUL (total time: 2 seconds)
```

12(cd). Widok serwera aplikacji po operacjach:

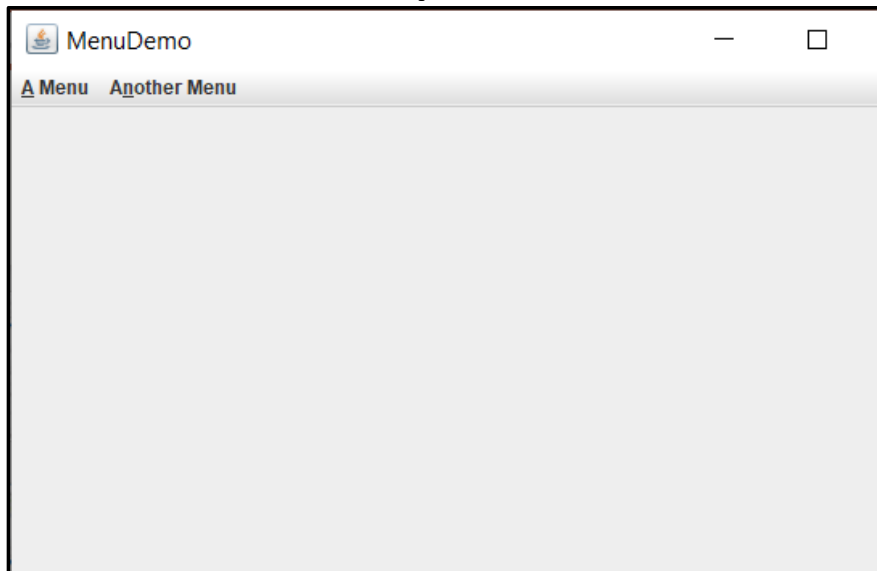
Deploy na aplikacji typu **Enterprise Application (Library1_EE)**, a potem **Run** na projekcie typu **Enterprise Application Client (Library1_EE_Client)**. Uruchomienie drugiej aplikacji **Library1_EE_Client** wykorzystuje ten sam kod na serwerze.

The screenshot displays an IDE interface with the following components:

- Navigator:** Shows a tree view of the project structure. Under "Servers" > "GlassFish Server5" > "Applications", the project "Library1_EE_Client" is selected. A red arrow points from the text above to this project.
- Source Editor:** Displays the code for "Client.java". The code includes imports for `businessstier.EJBFacadeRemote`, `javax.ejb.EJB`, and `library.Panel_util`. It defines a `Client` class with an `@EJB` annotation and a `getFacade()` method that returns the `facade` instance.
- Output Console:** Shows the execution output for "Library1_EE_Client (run)". The output includes the following text:

```
Redeploying C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\programy\library1\Library1_EE
post-run-deploy:
run-deploy:
Copying 1 file to C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\programy\library1\Libra
```
- Taskbar:** At the bottom, a taskbar entry shows "Library1_EE_Client (run) #2" with the status "running...".

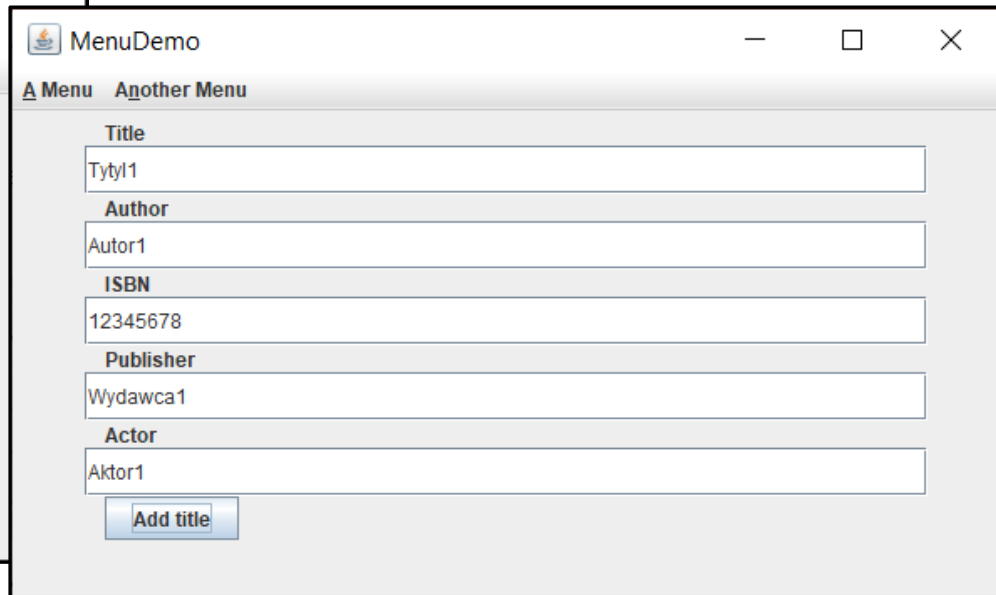
13. Formularze do wprowadzanie danych tytułów (z lewej) i książek (z prawej) – **uruchomienie dwóch aplikacji w *Warstwie klienta*** i wprowadzenie danych tytułów. Zostaną zapisane w *Warstwie biznesowej* w projekcie **Library_1IO**.



MenuDemo

A Menu Another Menu

This screenshot shows the MenuDemo application window with a menu bar containing 'A Menu' and 'Another Menu'. The main content area is empty.



MenuDemo

A Menu Another Menu

Title
Tytyl1

Author
Autor1

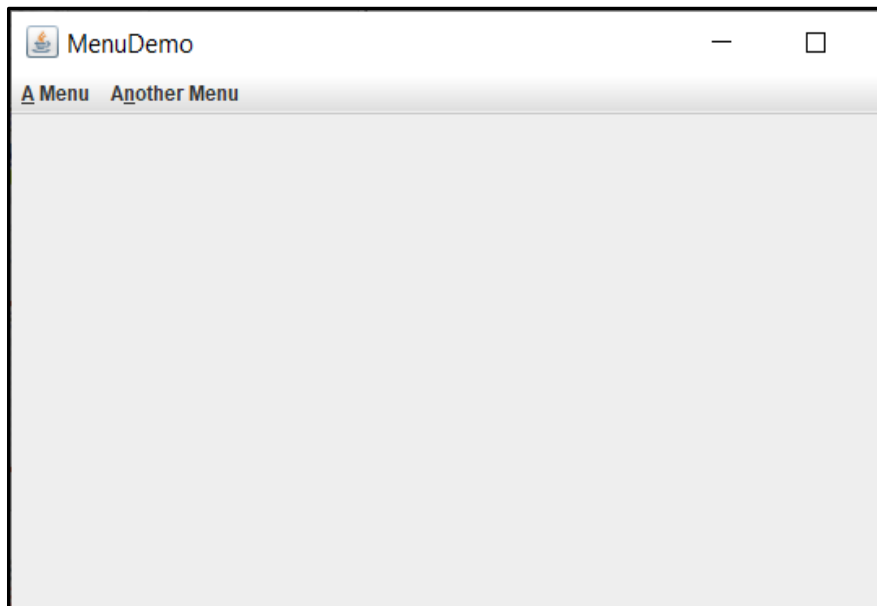
ISBN
12345678

Publisher
Wydawca1

Actor
Aktor1

Add title

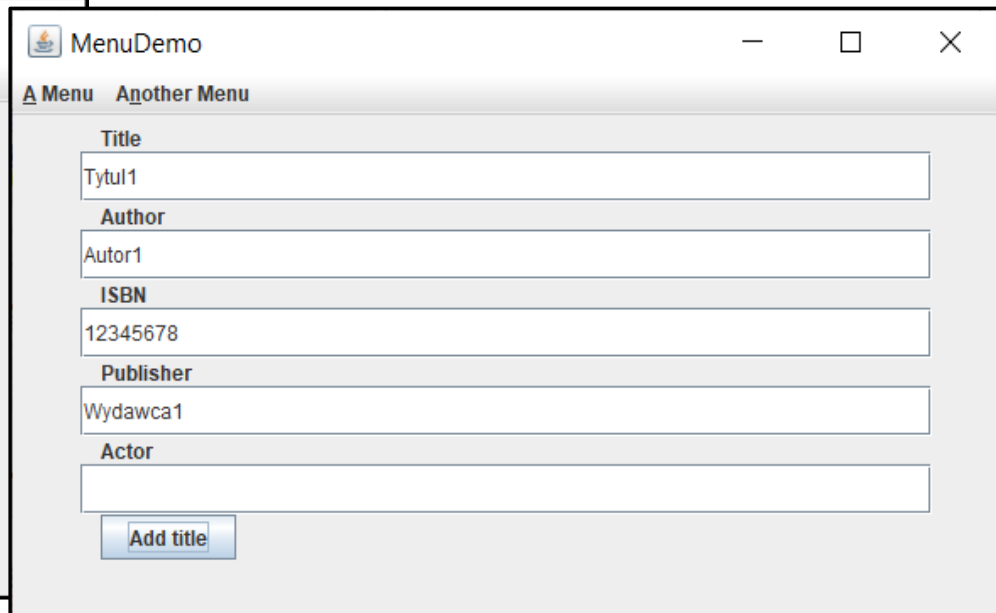
This screenshot shows the MenuDemo application window with a menu bar containing 'A Menu' and 'Another Menu'. The main content area contains a form with five text input fields: 'Title' (Tytyl1), 'Author' (Autor1), 'ISBN' (12345678), 'Publisher' (Wydawca1), and 'Actor' (Aktor1). Below the fields is an 'Add title' button.



MenuDemo

A Menu Another Menu

This screenshot shows the MenuDemo application window with a menu bar containing 'A Menu' and 'Another Menu'. The main content area is empty.



MenuDemo

A Menu Another Menu

Title
Tytul1

Author
Autor1

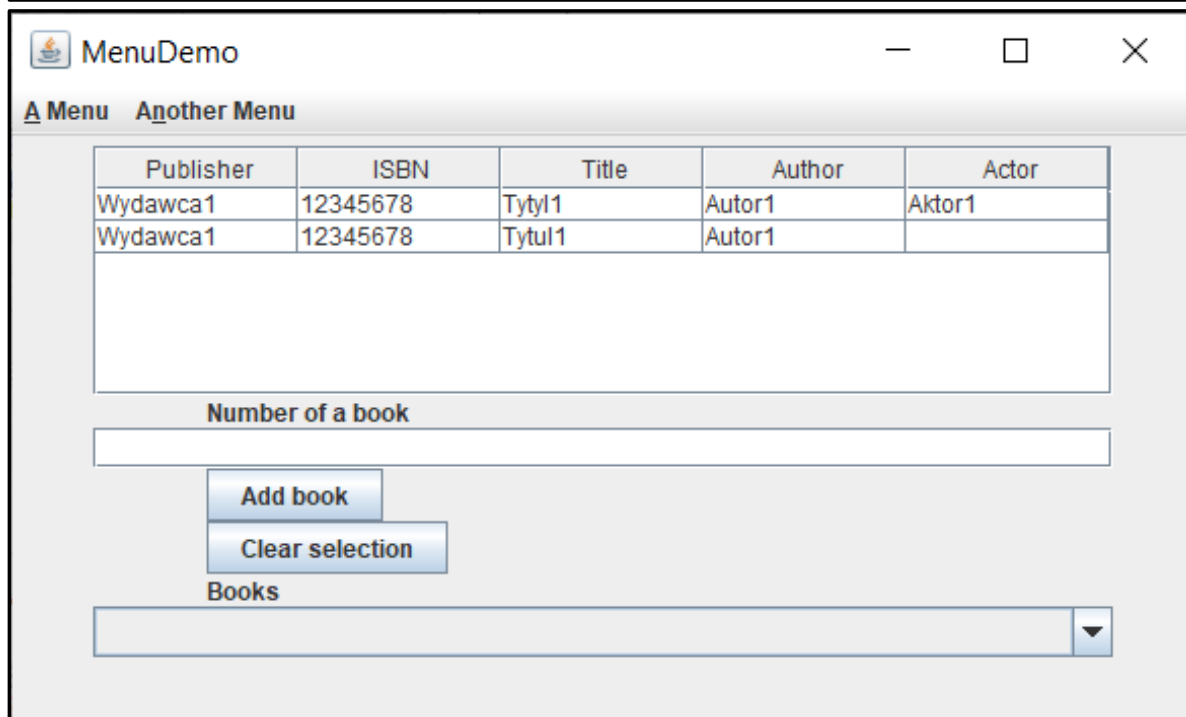
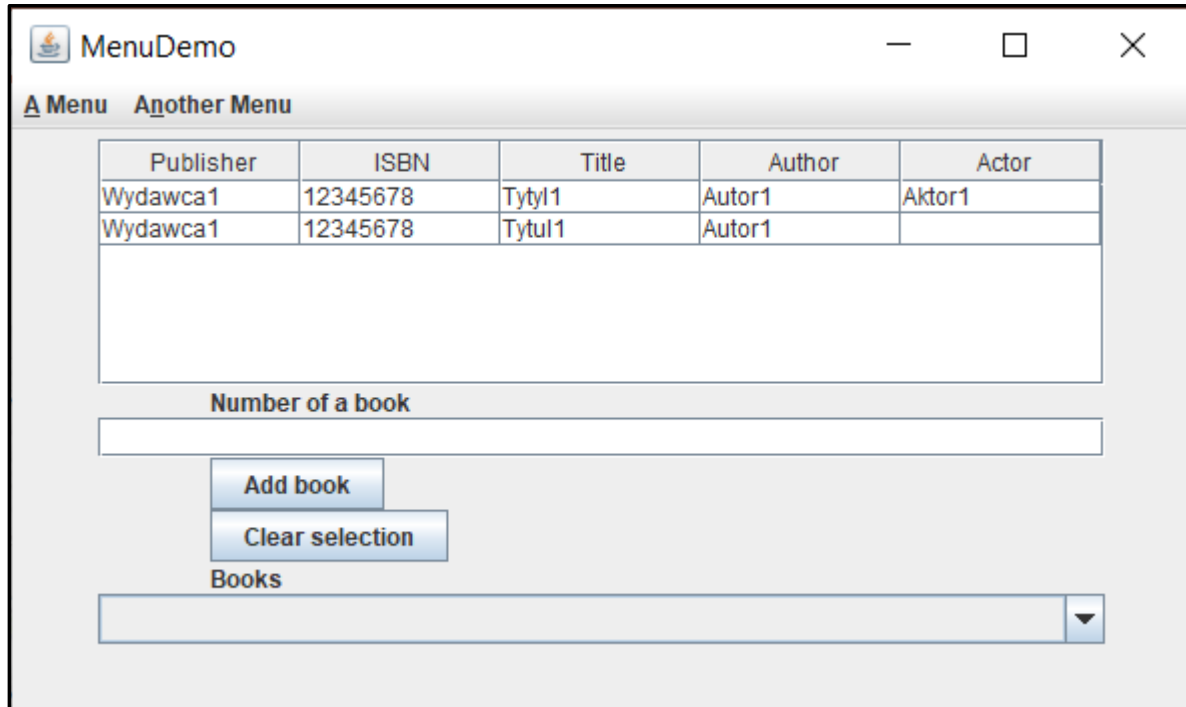
ISBN
12345678

Publisher
Wydawca1

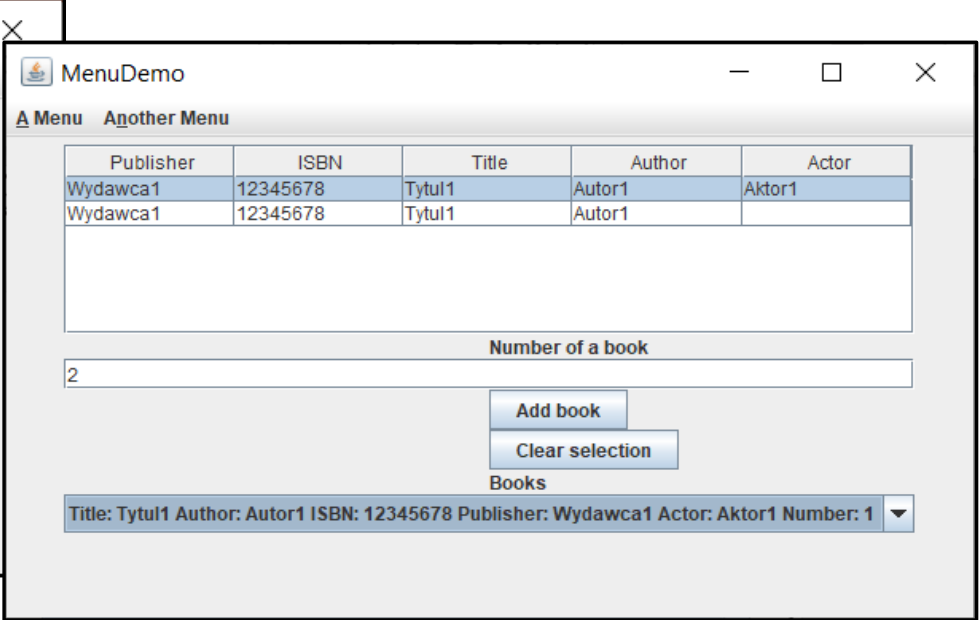
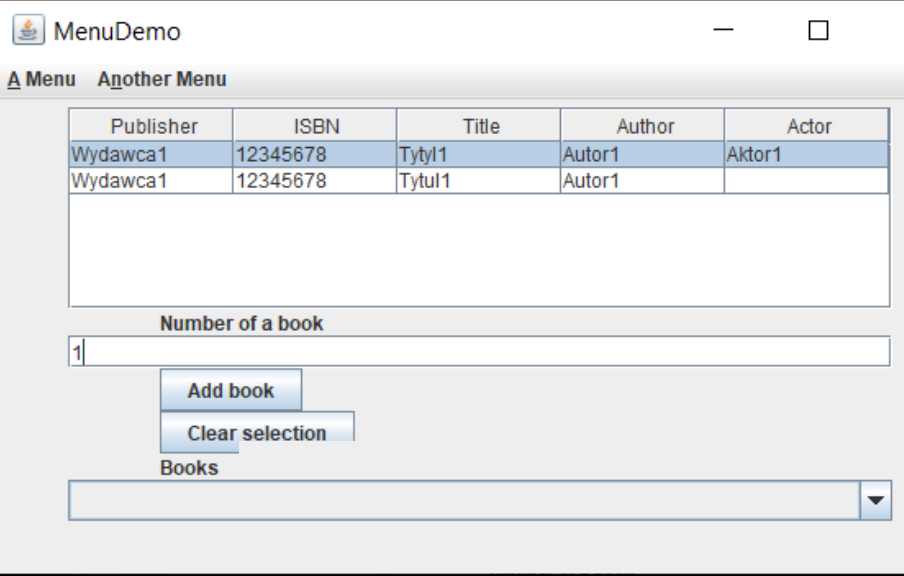
Actor

Add title

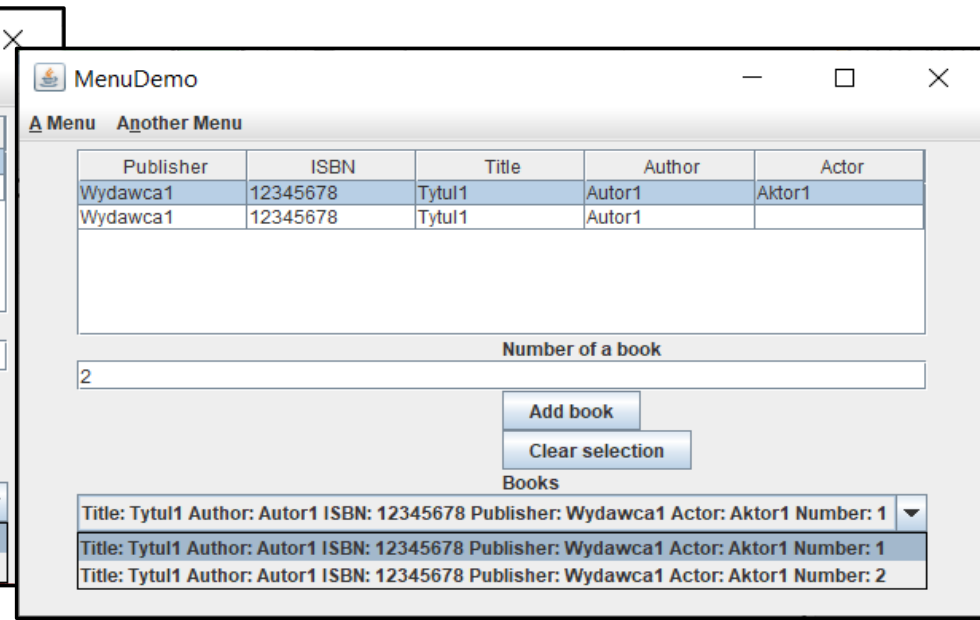
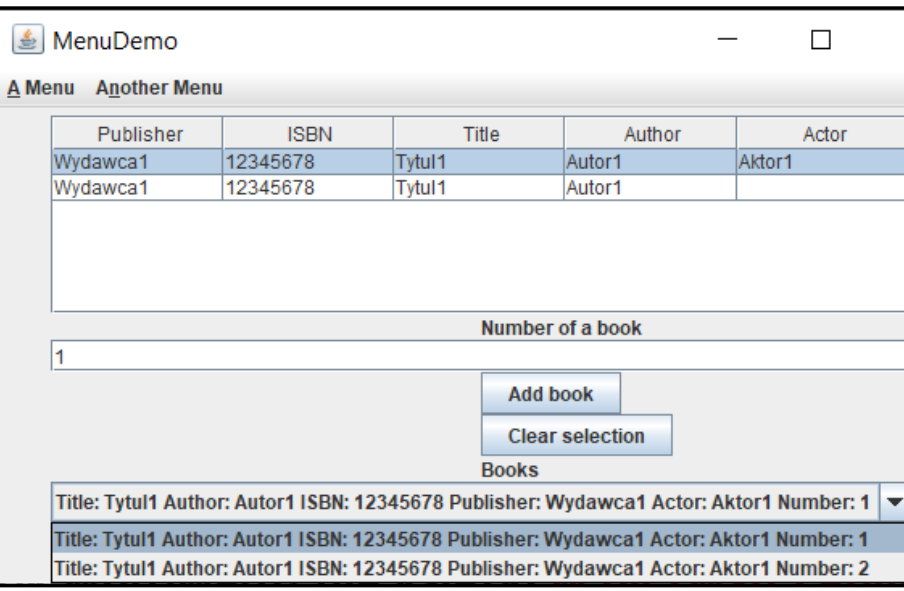
This screenshot shows the MenuDemo application window with a menu bar containing 'A Menu' and 'Another Menu'. The main content area contains a form with five text input fields: 'Title' (Tytul1), 'Author' (Autor1), 'ISBN' (12345678), 'Publisher' (Wydawca1), and 'Actor' (empty). Below the fields is an 'Add title' button.



14. Widok stron do wprowadzania książek w obu uruchomionych aplikacjach klienckich



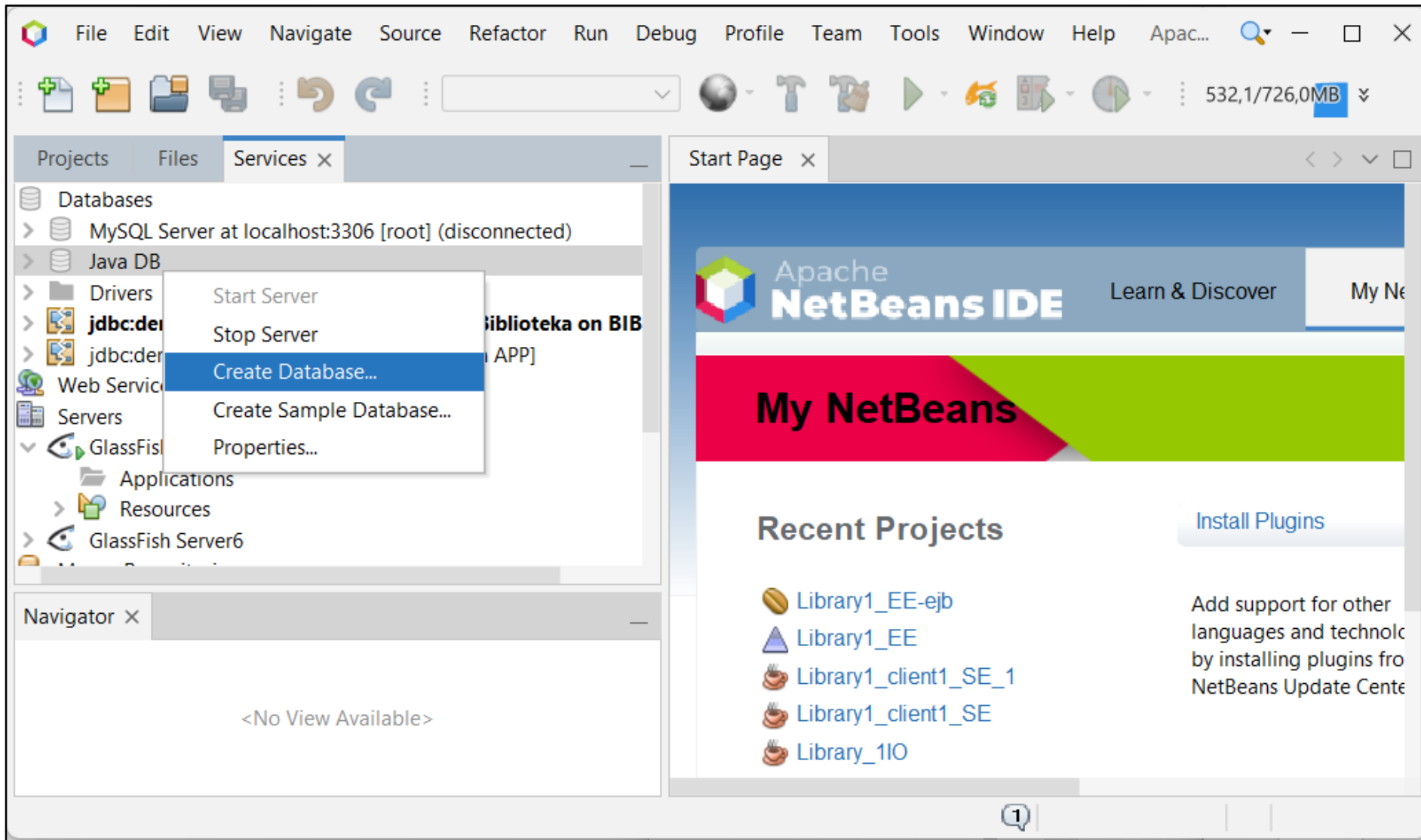
15. Widok wprowadzenia książek i widok wyniku ich wprowadzenia w obu uruchomionych aplikacjach klienckich



Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
6. Przykłady architektury wielowarstwowej aplikacji typu **EE** (p.2).
Wykonanie aplikacji typu **EE**.
Warstwa biznesowa: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów* (**EIS**)- baza danych w systemie baz danych
Apache Derby

1. Zakładanie pustej bazy danych w systemie **Apache Derby**



2. Zakładanie pustej bazy danych **Biblioteka** w systemie baz danych **Apache Derby**



Create Java DB Database

Database Name: Biblioteka

User Name: Biblioteka

Password: *****

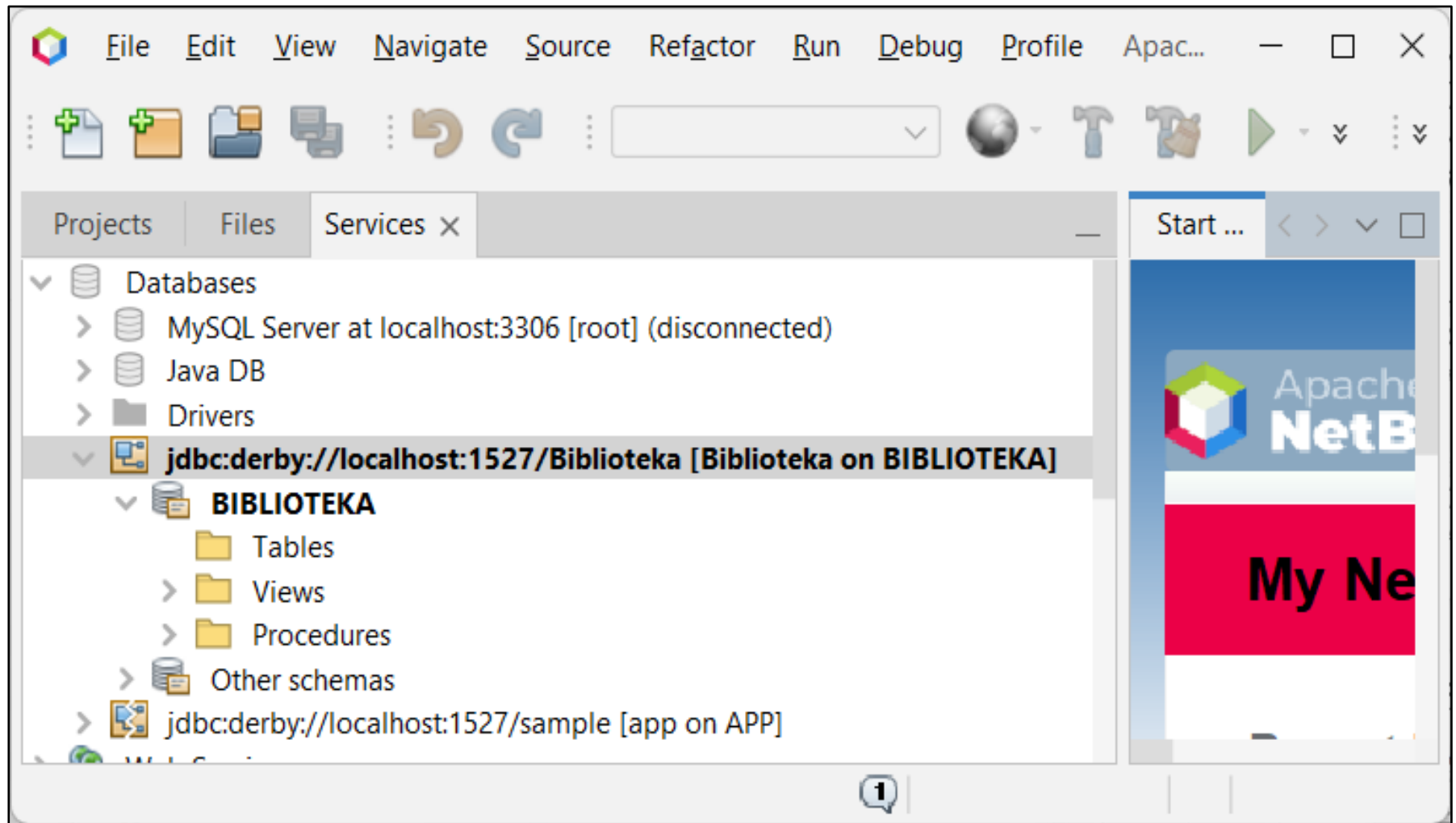
Confirm Password: *****

Database Location: C:\Derby

Properties...

OK Cancel

3. Utworzona pusta baza danych



Architektura oprogramowania

Pięciowarstwowy model logicznego rozdzielania zadań aplikacji
(wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

Warstwa klienta

Library1_EE_Client(client_tier, library)

Interakcja z użytkownikiem,
urządzenia i prezentacja
interfejsu użytkownika

Warstwa prezentacji

Logowanie, zarządzanie sesją,
tworzenie zawartości,
formatowania i dostarczanie

Warstwa biznesowa

Library_1IO, Library1_EE_Interface
Library1_EE, Library1_EE-ejb

Logika biznesowa, transakcje,
dane i usługi

Warstwa integracji

Adaptory zasobów, systemy
zewnętrzne, mechanizmy
zasobów, przepływ sterowania

Warstwa zasobów

Baza danych Biblioteka typu Apache Derby

Zasoby, dane i usługi
zewnętrzne

Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
6. Przykłady architektury wielowarstwowej aplikacji typu **EE** (p.2). Wykonanie aplikacji typu **EE** *Warstwa biznesowa*: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów (EIS)*- baza danych w systemie baz danych **Apache Derby**
8. **Utworzenie obiektowego modelu danych do utrwalania ORM**

```
@Entity
```

```
public class TitleBook implements Serializable {  
    private static final long serialVersionUID=1L;  
    private String publisher;  
    private String ISBN;  
    private String title;  
    private String author;  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    public Long getId() { ...3 lines }  
    public void setId(Long id) { ...3 lines }  
    @OneToMany(mappedBy = "titleBook", cascade=ALL)  
    List<Book> books;  
    public List<Book> getBooks() { ...3 lines }  
    public void setBooks(List<Book> books) { ...3 lines }  
    public TitleBook() { ...3 lines }  
  
    public String getPublisher() { ...3 lines }  
    public void setPublisher(String publisher) { ...3 lines }  
    public String getISBN() { ...3 lines }  
    public void setISBN(String ISBN) { ...3 lines }  
    public String getTitle() { ...3 lines }  
    public void setTitle(String title) { ...3 lines }  
    public String getAuthor() { ...3 lines }  
    public void setAuthor(String author) { ...3 lines }  
    public String getActor() { ...3 lines }  
    public void setActor(String val) { ...2 lines }
```

Zmiana typu klas danych na typ **@Entity** w projekcie **Warstwy biznesowej** – dodano adnotacje, nowy atrybut **Id**. Należy zestandaryzować nazwy metod dostępu do składowych klasy typu **Entity**.

```
package subbusinessstier.entities;

+ import ...

@Entity
public class TitleBookRead extends TitleBook {
private static final long serialVersionUID=1L;
    private String actor;

    @Override
+ public String getActor() {...3 lines }

    @Override
+ public void setActor(String actor) {...3 lines }

    @Override
+ public String toString() {...5 lines }

}
```

```
@Entity
```

```
public class Book implements Serializable {
```

```
private static final long serialVersionUID=1L;
```

```
private int number;
```

```
@ManyToOne
```

```
private TitleBook titleBook;
```

```
// @OneToMany (mappedBy="book")
```

```
@Transient
```

```
private List<Reservation> reservations;
```

```
@Id
```

```
@GeneratedValue(strategy = GenerationType.AUTO)
```

```
private Long id;
```

```
public Long getId() { ...3 lines }
```

```
public void setId(Long id) { ...3 lines }
```

```
public List<Reservation> getReservations() { ...3 lines }
```

```
public void setReservations(List<Reservation> reservations) { ...3 lines }
```


Zagadnienia

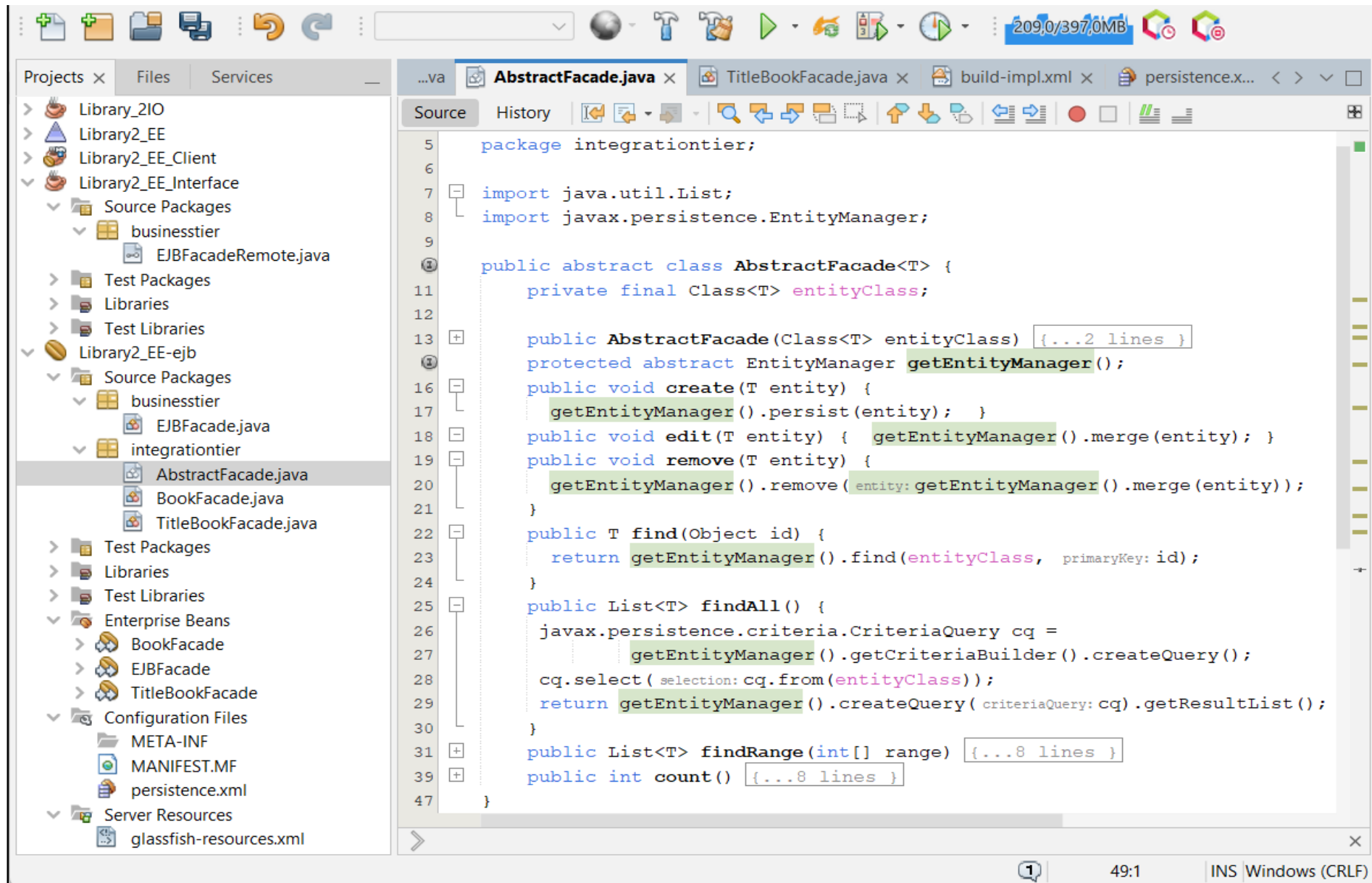
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
6. Przykłady architektury wielowarstwowej aplikacji typu **EE** (p.2). Wykonanie aplikacji typu **EE**. *Warstwa biznesowa*: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów (EIS)*- baza danych w systemie baz danych **Apache Derby**
8. Utworzenie obiektowego modelu danych do utrwalania **ORM**
9. *Warstwa integracji*. Zastosowanie wzorca projektowego typu *Domain Store* w technologii **JPA (Java Persistence)** na platformie **Java EE**

Architektura oprogramowania

Pięciowarstwowy model logicznego rozdzielania zadań aplikacji
(wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)



1. Klasa abstrakcyjna generyczna **AbstractFacade** z definicją uniwersalnych metod obsługujących transakcje **JPA** - parametr **T** może być zastąpiony każdą z klas typu **Entity**



```
5 package integrationtier;
6
7 import java.util.List;
8 import javax.persistence.EntityManager;
9
10 public abstract class AbstractFacade<T> {
11     private final Class<T> entityClass;
12
13     public AbstractFacade(Class<T> entityClass) {...2 lines }
14     protected abstract EntityManager getEntityManager();
15
16     public void create(T entity) {
17         getEntityManager().persist(entity); }
18     public void edit(T entity) { getEntityManager().merge(entity); }
19     public void remove(T entity) {
20         getEntityManager().remove(entity);
21         getEntityManager().merge(entity); }
22     public T find(Object id) {
23         return getEntityManager().find(entityClass, primaryKey: id);
24     }
25     public List<T> findAll() {
26         javax.persistence.criteria.CriteriaQuery cq =
27             getEntityManager().getCriteriaBuilder().createQuery();
28         cq.select(selection: cq.from(entityClass));
29         return getEntityManager().createQuery(criteriaQuery: cq).getResultList();
30     }
31     public List<T> findRange(int[] range) {...8 lines }
32
33     public int count() {...8 lines }
34
35 }
```

The screenshot shows an IDE window with the following content:

- Projects:** Library_2IO, Library2_EE, Library2_EE_Client, Library2_EE_Interface, Library2_EE-ejb, Enterprise Beans, Configuration Files, Server Resources.
- Files:** Source Packages, Test Packages, Libraries, Test Libraries.
- Services:** ...va, AbstractFacade.java, TitleBookFacade.java, build-impl.xml, persistence.x...
- Code Editor:** Source view of AbstractFacade.java with line numbers 5 to 47. The code defines an abstract generic class with methods for create, edit, remove, find, findAll, findRange, and count, all using an EntityManager.

2. Klasa **TitleBookFacade** implementująca klasę **AbstractFacade**

- kontroler typu **Session Bean** do utrwalania obiektów typu **TitleBook** i **TitleBookRead**

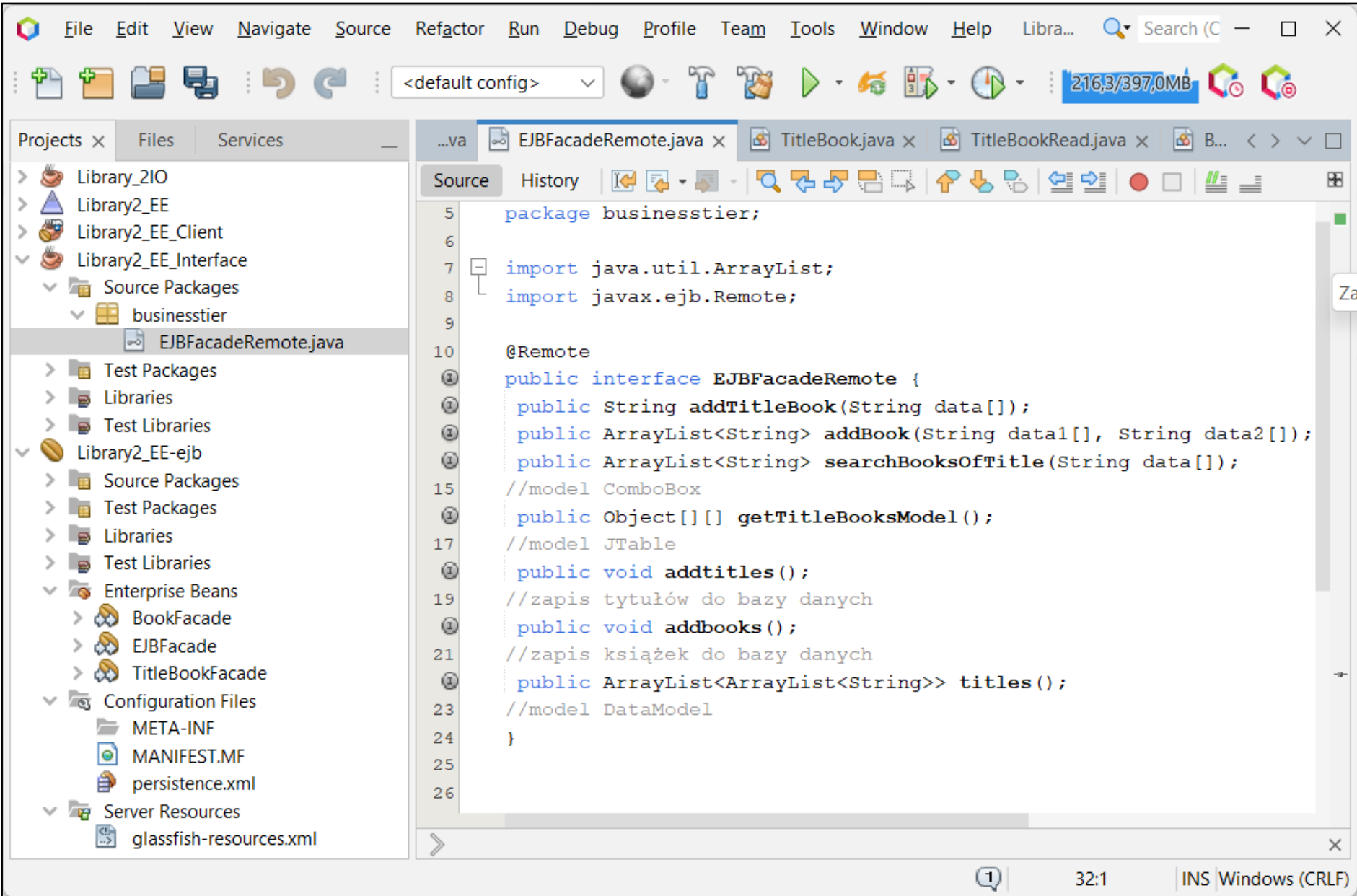
```
5 package integrationtier;
6
7 import java.util.List;
8 import javax.ejb.Stateless;
9 import javax.persistence.EntityManager;
10 import javax.persistence.PersistenceContext;
11 import subbusinessstier.entities.TitleBook;
12
13 @Stateless
14 public class TitleBookFacade extends AbstractFacade<TitleBook> {
15     @PersistenceContext(unitName = "Library2_EE-ejbPU")
16     private EntityManager em;
17     @Override
18     protected EntityManager getEntityManager() { return em; }
19     public TitleBookFacade() { super(entityClass: TitleBook.class); }
20     public void addTitleBooks(List<TitleBook> titles) {
21         for(TitleBook title: titles)
22             if (title.getId() == null)
23                 getEntityManager().persist(entity: title);
24     }
25 }
26
27
28
29
30
31
32
```

34:1 | INS Windows (CRLF)

3. Klasa **BookFacade** implementująca klasę **AbstractFacade** - kontroler typu **Session Bean** do utrwalania obiektów typu **Book**

```
5 package integrationtier;
6
7 import java.util.List;
8 import javax.ejb.Stateless;
9 import javax.persistence.EntityManager;
10 import javax.persistence.PersistenceContext;
11 import subbusinesstier.entities.Book;
12 import subbusinesstier.entities.TitleBook;
13
14 @Stateless
15 public class BookFacade extends AbstractFacade<Book> {
16     @PersistenceContext(unitName = "Library2_EE-ejbPU")
17     private EntityManager em;
18
19     @Override
20     protected EntityManager getEntityManager() { return em; }
21     public BookFacade() { super(entityClass: Book.class); }
22     public void addBooks(List<TitleBook> titles) {
23         for (TitleBook title:titles) {
24             if (title.getId() == null)
25                 continue;
26             for(Book book:title.getBooks())
27                 if (book.getId() == null)
28                     getEntityManager().persist(entity:book);
29         }
30     }
31 }
32
```

5. Uzupełnienie deklaracji metod o zdalnym dostępie do metod komponentów do utrwalania obiektów typu **Entity** w interfejsie komponentu typu **Session Bean**



The screenshot shows an IDE window with the following components:

- Top Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, Libra... Search (C - □ X)
- Toolbar:** <default config>, various icons for file operations and execution.
- Project Explorer (Left):** Library_2IO, Library2_EE, Library2_EE_Client, Library2_EE_Interface (expanded), Source Packages (expanded), businessstier (expanded), EJBFacadeRemote.java (selected), Test Packages, Libraries, Test Libraries, Library2_EE-ejb (expanded), Source Packages, Test Packages, Libraries, Test Libraries, Enterprise Beans (expanded), BookFacade, EJBFacade, TitleBookFacade, Configuration Files (expanded), META-INF, MANIFEST.MF, persistence.xml, Server Resources (expanded), glassfish-resources.xml.
- Source Editor (Center):** EJBFacadeRemote.java

```
5 package businessstier;
6
7 import java.util.ArrayList;
8 import javax.ejb.Remote;
9
10 @Remote
11 public interface EJBFacadeRemote {
12     public String addTitleBook(String data[]);
13     public ArrayList<String> addBook(String data1[], String data2[]);
14     public ArrayList<String> searchBooksOfTitle(String data[]);
15     //model ComboBox
16     public Object[][] getTitleBooksModel();
17     //model JTable
18     public void addtitles();
19     //zapis tytułów do bazy danych
20     public void addbooks();
21     //zapis książek do bazy danych
22     public ArrayList<ArrayList<String>> titles();
23     //model DataModel
24 }
25
26
```
- Bottom Bar:** 1 | 32:1 | INS Windows (CRLF)

4. Klasa **EJBFacade** - główny komponent *Warstwy biznesowej* udostępnia **metody logiki biznesowej** i zarządza komponentami do utrwalania obiektów typu **Entity**.

```
16 @Stateless
17 public class EJBFacade implements EJBFacadeRemote {
18     @EJB
19     private BookFacade bookFacade;
20     @EJB
21     private TitleBookFacade titleBookFacade;
22
23     Facade facade = new Facade();
24     @Override
25     public String addTitleBook(String data[]) {return facade.addTitleBook(data); }
26     @Override
27     public ArrayList<String> addBook(String data1[], String data2[]) {return facade.addBook(data1, data2);}
28     @Override
29     public ArrayList<String> searchBooksOfTitle(String data[]) {return facade.searchBooksOfTitle(data);}
30     @Override
31     public Object[][] getTitleBooksModel() {return facade.getTitleBooksModel();}
32
33     @Override
34     public void addtitles () {titleBookFacade.addTitleBooks ( titles: facade.getTitleBooks ());}
35     @Override
36     public void addbooks () { bookFacade.addBooks ( titles: facade.getTitleBooks ()); }
37     @Override
38     public ArrayList<ArrayList<String>> titles () {
39         List<TitleBook> help1 = titleBookFacade.findAll ();
40         ArrayList<ArrayList<String>> help2 = new ArrayList<> ();
41         for (TitleBook t : help1) {
42             ArrayList<String> help3 = new ArrayList<> ();
43             help3.add ( e: t.getPublisher ());
44             help3.add ( e: t.getISBN ());
45             help3.add ( e: t.getTitle ());
46             help3.add ( e: t.getAuthor ());
47             help3.add ( e: t.getActor ());
48             help2.add ( e: help3); }
49         return help2; }
50 }
```


5. Uruchomienie desktopowej aplikacji wielowarstwowej typu EE

The screenshot shows an IDE interface with a context menu open over a project folder. The 'Deploy' option is highlighted with a red rectangle. The background displays the source code of `EJBFacade.java` and a terminal window showing the build process.

```
@Stateless
public class EJBFacade implements EJBFacadeRemote {
    @EJB
    private BookFacade bookFacade;
    @EJB
    private TitleBookFacade titleBookFacade;

    Facade facade = new Facade();
    @Override
    public String addTitleBook(String data[]) {return facade.addTitleBook(data); }
    @Override
    public ArrayList<String> addBook(String data1[], String data2[]) {return facade.addBook(data1, data2);}
    @Override
    public ArrayList<String> searchBooksOfTitle(String data[]) {return facade.searchBooksOfTitle(data);}
    @Override
    public Object[][] getTitleBooksModel ()
        {return facade.getTitleBooksModel();}

    @Override
    public void addtitles ()
        {titleBookFacade.addTitleBooks ( titles: facade.getTitleBooks());}
    @Override
    public void addbooks ()
        { bookFacade.addBooks ( titles: facade.getTitleBooks()); }
    @Override
    public ArrayList<ArrayList<String>> titles () {
        List<TitleBook> help1 = titleBookFacade.findAll();
        ArrayList<ArrayList<String>> help2 = new ArrayList<>();
```

```
- Library2_EE (clean,dist) x
pre-dist:
do-dist-without-manifest:
do-dist-with-manifest:
Created dir: C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\programy\library2\Library2_EE\dist
Building jar: C:\Studia2021_2022\IO\IO_INEK1\Wyklady\wyklad6\2022_2023\programy\library2\Library2_EE\dist\Libr
post-dist:
dist:
BUILD SUCCESSFUL (total time: 3 seconds)
```

6. Wygenerowanie pustych tabel w bazie danych

The screenshot shows an IDE window with the following components:

- Left Panel (Database Explorer):** Shows a tree view of databases. The selected database is `jdbc:derby://localhost:1527/Biblioteka [Biblioteka on BIBLIOTEKA]`. Underneath, the `BIBLIOTEKA` schema is expanded to show a `TITLEBOOK` table. The table columns are: `ID`, `DTYPE`, `ISBN`, `AUTHOR`, `PUBLISHER`, `TITLE`, and `ACTOR`.
- Right Panel (Code Editor):** Shows the source code for `EJBFacade.java`. The code includes imports for `integrationtier`, `java.util`, and `javax.ejb`. It defines a `@Stateless` class `EJBFacade` that implements `EJBFacadeRemote`. The class has two private attributes: `BookFacade bookFacade` and `TitleBookFacade titleBookFacade`.
- Bottom Panel (Output Console):** Shows the output of the application. It includes a warning about a deprecated method in `java.lang.System` and a message indicating that the Apache Derby server is running.

```
package businesstier;

import integrationtier.BookFacade;
import integrationtier.TitleBookFacade;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import subbusinesstier.Facade;
import subbusinesstier.entities.TitleBook;

@Stateless
public class EJBFacade implements EJBFacadeRemote {
    @EJB
    private BookFacade bookFacade;
    @EJB
    private TitleBookFacade titleBookFacade;
}
```

Output Console:

```
Sun Sep 18 13:16:17 CEST 2022 : Security manager installed using the Basic server security policy
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by org.apache.derby.drda.NetworkServerControl
WARNING: Please consider reporting this to the maintainers of org.apache.derby.drda.NetworkServerControl
WARNING: System::setSecurityManager will be removed in a future release
Sun Sep 18 13:16:17 CEST 2022 : Serwer sieciowy Apache Derby - 10.14.2.0 - (1828579) uruchomiony :
```

Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej.
6. Przykłady architektury wielowarstwowej aplikacji typu **EE** (p.2). Wykonanie aplikacji typu **EE**. *Warstwa biznesowa*: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów (EIS)*- baza danych w systemie baz danych **Apache Derby**
8. Utworzenie obiektowego modelu danych do utrwalania **ORM**
9. *Warstwa integracji*. Zastosowanie wzorca projektowego typu *Domain Store* w technologii **JPA (Java Persistence)** na platformie **Java EE**
10. *Warstwa prezentacji* – **JSF** – dodanie stron www do komunikacji z bazą danych

Architektura oprogramowania

Pięciowarstwowy model logicznego rozdzielania zadań aplikacji
(wg. D.Alur, J.Crupi, D. Malks, Core J2EE. Wzorce projektowe.)

Warstwa klienta

Library2_EE_Client (client_tier, library)

Interakcja z użytkownikiem,
urządzenia i prezentacja
interfejsu użytkownika

Warstwa prezentacji

Library2_EE_WEB

Logowanie, zarządzanie sesją,
tworzenie zawartości,
formatowania i dostarczanie

Warstwa biznesowa

Library_2IO, Library2_EE_Interface
Library2_EE, Library2_EE-ejb(businessstier)

Logika biznesowa, transakcje,
dane i usługi

Warstwa integracji

Library2_EE-ejb (integrationtier)

Adaptory zasobów, systemy
zewnętrzne, mechanizmy
zasobów, przepływ sterowania

Warstwa zasobów

Baza danych Biblioteka typu Apache Derby

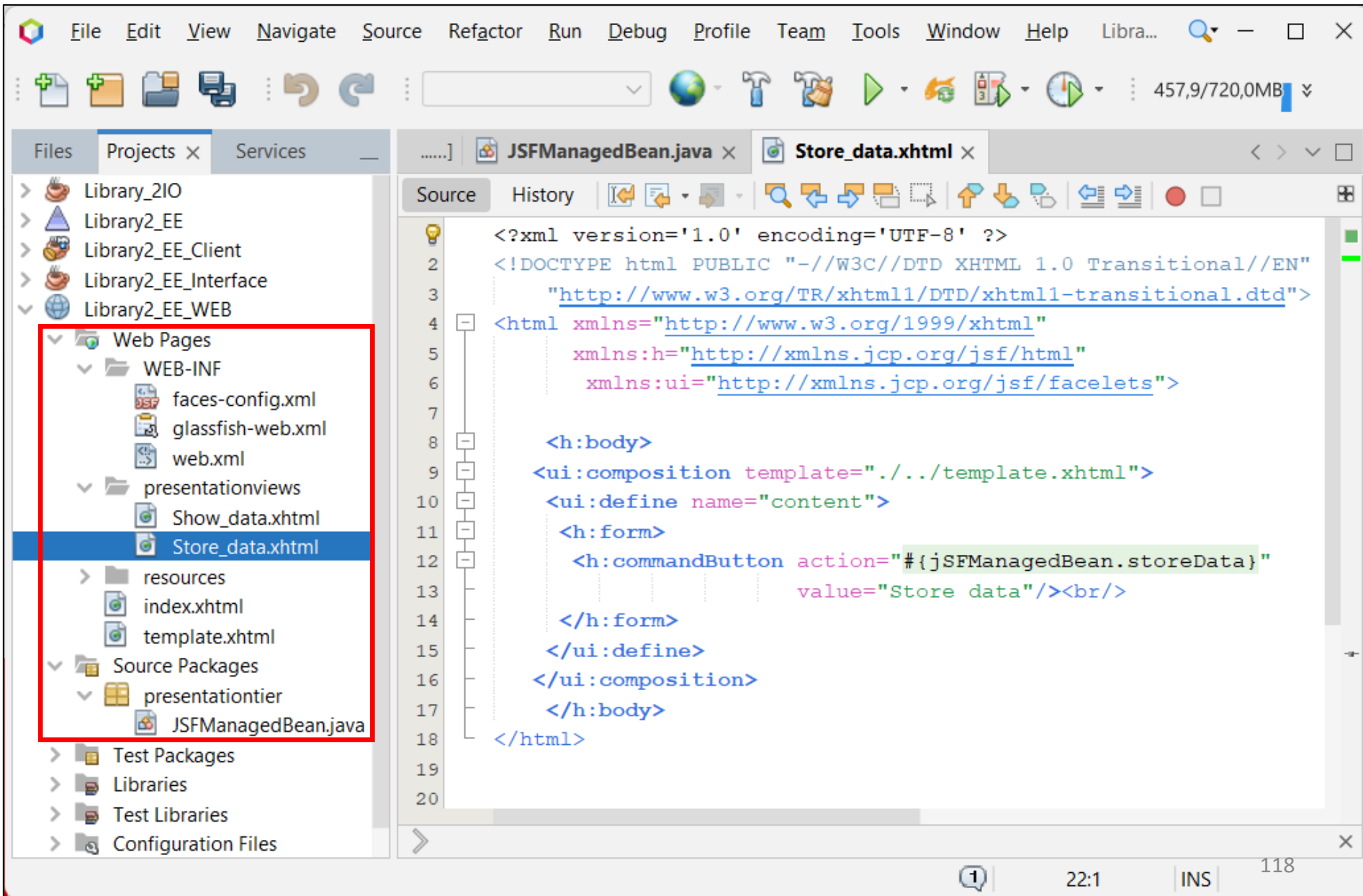
Zasoby, dane i usługi
zewnętrzne

The screenshot shows an IDE window with a project explorer on the left and a source code editor on the right. The project explorer shows a project named 'Library2_EE_WEB' with a 'Web Pages' folder containing 'Show_data.xhtml' and 'Store_data.xhtml'. The source code editor displays the XML code for 'Show_data.xhtml', which includes a JSF form and a data table. A red box highlights the 'Web Pages' folder in the project explorer. A white box with a black border contains a numbered list item.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:body>
<ui:composition template="../../template.xhtml">
<ui:define name="content">
<h:form styleClass="jsfcrud_list_form">
<h:panelGroup id="messagePanel" layout="block">
<h:messages errorStyle="color: red" infoStyle="color: green" layout="table"/>
</h:panelGroup>
<h:outputText escape="false" value="Lista produktow pusta"
rendered="#{jSFManagedBean.items.rowCount == 0}"/>
<h:panelGroup rendered="#{jSFManagedBean.items.rowCount > 0}">
<h:dataTable value="#{jSFManagedBean.items}" var="item" border="0"
cellpadding="2" cellspacing="0" rowClasses="jsfcrud_odd_row,jsfcrud_even_row"
rules="all" style="border:solid 1px">
<h:column> <f:facet name="header"> <h:outputText value="Publisher"/></f:facet>
<h:outputText value="#{item.get(0)}"/> </h:column>
<h:column> <f:facet name="header"> <h:outputText value="ISBN"/> </f:facet>
<h:outputText value="#{item.get(1)}"/> </h:column>
<h:column> <f:facet name="header"> <h:outputText value="Title"/> </f:facet>
<h:outputText value="#{item.get(2)}"/> </h:column>
<h:column> <f:facet name="header"> <h:outputText value="Author"/> </f:facet>
<h:outputText value="#{item.get(3)}"/> </h:column>
<h:column> <f:facet name="header"> <h:outputText value="Actor"/> </f:facet>
<h:outputText value="#{item.get(4)}"/> </h:column>
</h:dataTable>
</h:panelGroup>
</h:form>
</ui:define>
</ui:composition>
</h:body>
</html>
```

1. Wykonanie projektu typu **Web Application Library2_EE_WEB** i utworzenie formularzy **JSF**. Formularz **Show_data.xhtml** wyświetla dane tytułów książek w formie tabeli.

2. Definiowanie formularzy JSF (cd) – utworzenie formularza `Store_data.xhtml` umożliwiającego zapis danych tytułów w bazie danych



The screenshot displays an IDE window with the following components:

- Top Menu:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help, Libra...
- Toolbar:** Includes icons for file operations, navigation, and execution. Memory usage is shown as 457,9/720,0MB.
- Project Explorer (Left):** Shows a project structure with folders like Library_2IO, Library2_EE, and Library2_EE_WEB. A red box highlights the `Web Pages` folder, which contains `WEB-INF` (with `faces-config.xml`, `glassfish-web.xml`, and `web.xml`), `presentationviews` (with `Show_data.xhtml` and `Store_data.xhtml`), `resources` (with `index.xhtml` and `template.xhtml`), and `Source Packages` (with `presentationtier` and `JSFManagedBean.java`).
- Editor (Right):** Shows the content of `Store_data.xhtml` with the following XML code:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <h:body>
    <ui:composition template="../../template.xhtml">
      <ui:define name="content">
        <h:form>
          <h:commandButton action="#{jSFManagedBean.storeData}"
            value="Store data"/><br/>
        </h:form>
      </ui:define>
    </ui:composition>
  </h:body>
</html>
```

The status bar at the bottom shows the cursor position at 22:1, the mode is INS, and the page number is 118.

3. Definicja komponentu typu **Managed Bean** – kontrolera widoków w JSF

Dostęp do logiki biznesowej za pomocą wzorca **SessionFacade**, występującego również w roli wzorca **Singleton** (komponent EJB)

Files Projects x Services

- Library_2IO
- Library2_EE
- Library2_EE_Client
- Library2_EE_Interface
- Library2_EE_WEB
 - Web Pages
 - WEB-INF
 - faces-config.xml
 - glassfish-web.xml
 - web.xml
 - presentationviews**
 - Show_data.xhtml
 - Store_data.xhtml
 - resources
 - index.xhtml
 - template.xhtml
 - Source Packages
 - presentationtier
 - JSFManagedBean.java
 - Test Packages
 - Libraries
 - Library2_EE_Interface - dist/Library2_EE_Interface.jar
 - JSF 2.3 - javax.faces.jar
 - JDK 1.8
 - GlassFish Server5
 - Test Libraries
 - Configuration Files
 - MANIFEST.MF
 - faces-config.xml
 - glassfish-web.xml
 - web.xml
 - Library2_EE-ejb

```
5 package presentationtier;
6
7 import business-tier.EJBFacadeRemote;
8 import javax.inject.Named;
9 import javax.ejb.EJB;
10 import javax.enterprise.context.RequestScoped;
11 import javax.faces.model.DataModel;
12 import javax.faces.model.ListDataModel;
13
14 @Named(value = "jSFManagedBean")
15 @RequestScoped
16 public class JSFManagedBean{
17
18     @EJB
19     private EJBFacadeRemote eJBFacade;
20
21     private DataModel items;
22
23     public JSFManagedBean() { }
24
25     public String storeData() {
26         eJBFacade.addtitles();
27         eJBFacade.addbooks();
28         return "/faces/index";
29     }
30     public String showData() {
31         createDataModel();
32         return "/faces/presentationviews/Show_data";
33     }
34 }
```

presentationtier.JSFManagedBean > getItems >

MenuDemo

A Menu Another Menu

Title
Tytul1

Author
Autor1

ISBN
123456789

Publisher
Wydawca1

Actor
Aktor1

Add title

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	123456789	Tytul1	Autor1	Aktor1
Wydawca1	123456789	Tytul1	Autor1	

Number of a book
1

Add book

Clear selection

Books

4. Uruchomienie bazodanowej aplikacji typu Enterprise **Library2_EE** na platformie JavaEE za pomocą operacji **Deploy** zawierającej dwa typy aplikacji klienckiej: desktopową **Library2_EE_Client** i internetową **Library2_EE_WEB**.

Uruchomiono 2 instancje aplikacji desktopowej za pomocą operacji **Run** oraz jedną instancję aplikacji internetowej również za pomocą operacji **Run**.

1-aplikacja desktopowa

MenuDemo

A Menu Another Menu

Title

Tytul1

Author

Autor1

ISBN

123456789

Publisher

Wydawca1

Actor

Add title

4 (cd).
 Uruchomiona 2-a
 aplikacja desktopowa

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	123456789	Tytul1	Autor1	Aktor1
Wydawca1	123456789	Tytul1	Autor1	

Number of a book

2

Add book

Clear selection

Books

Title: Tytul1 Author: Autor1 ISBN: 123456789 Publisher: Wydawca1 Actor: Aktor1 Number: 1

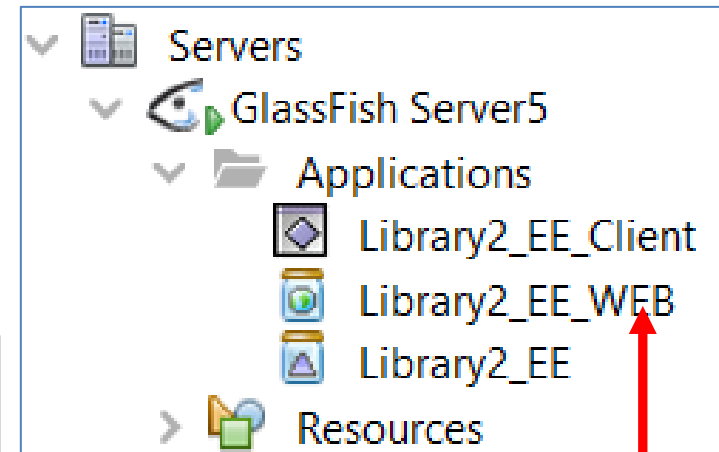
Title: Tytul1 Author: Autor1 ISBN: 123456789 Publisher: Wydawca1 Actor: Aktor1 Number: 1

Title: Tytul1 Author: Autor1 ISBN: 123456789 Publisher: Wydawca1 Actor: Aktor1 Number: 2

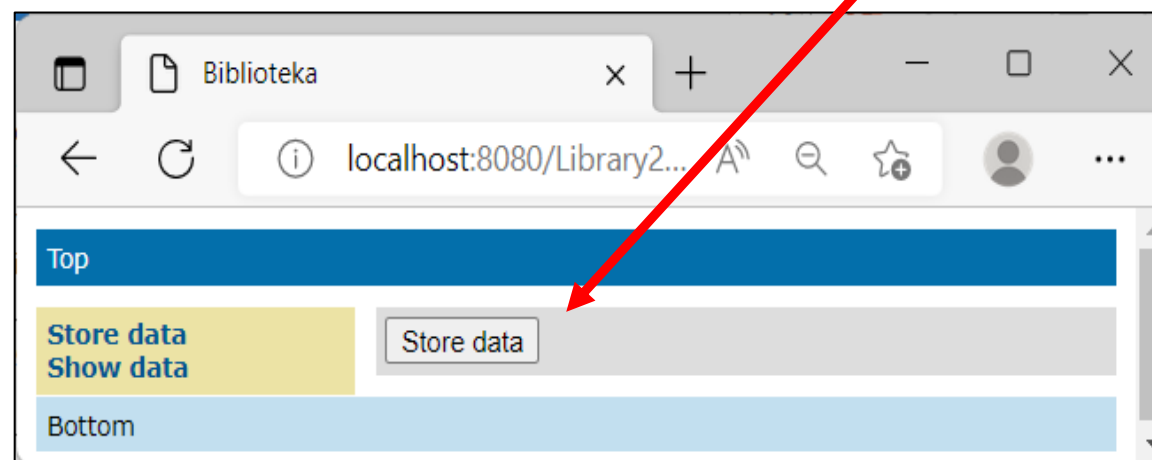
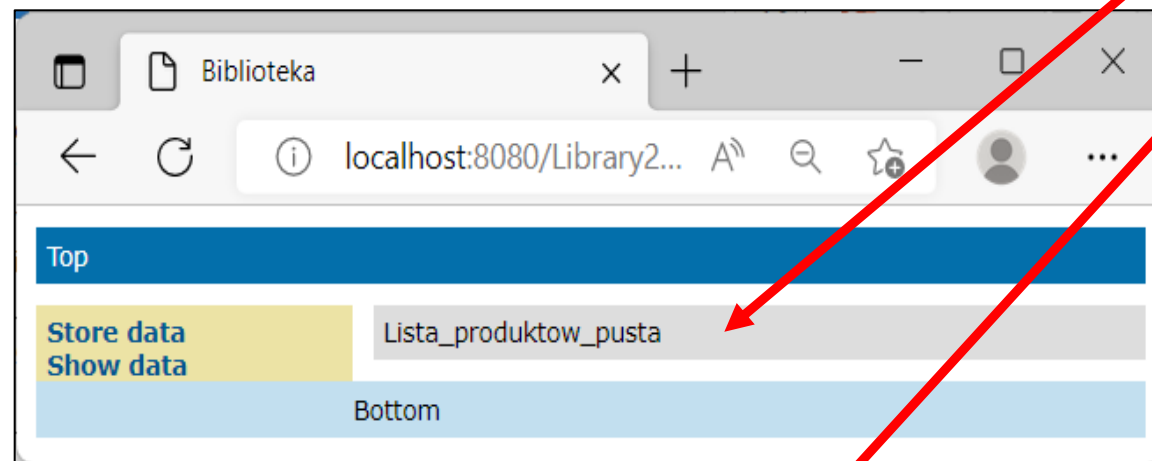
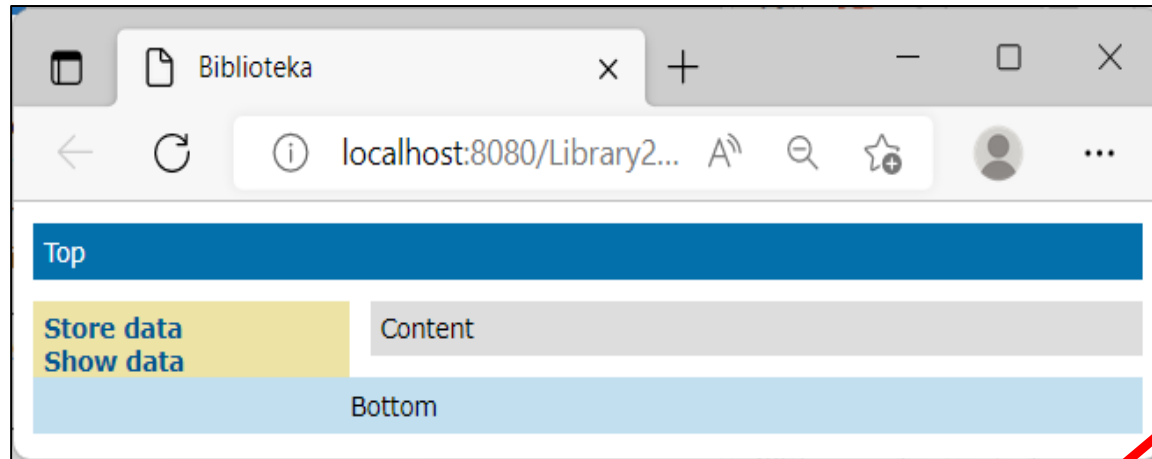
5. Uruchomiona 1-a aplikacja internetowa za pomocą operacji **Run**.

Formularz **Show data** wyświetla tytuły książek zapisanych w bazie danych - teraz baza danych jest pusta.

Formularz **Store data** służy do zapisania danych tytułów i książek przechowywanych w **Warstwie biznesowej** (projekt **Library_2IO**) do bazy danych



Po uruchomieniu 1-ej aplikacji internetowej **Library2_EJB_WEB** na serwerze utworzył się kod tej aplikacji używany dla kolejno uruchomionych aplikacji internetowych



Zagadnienia

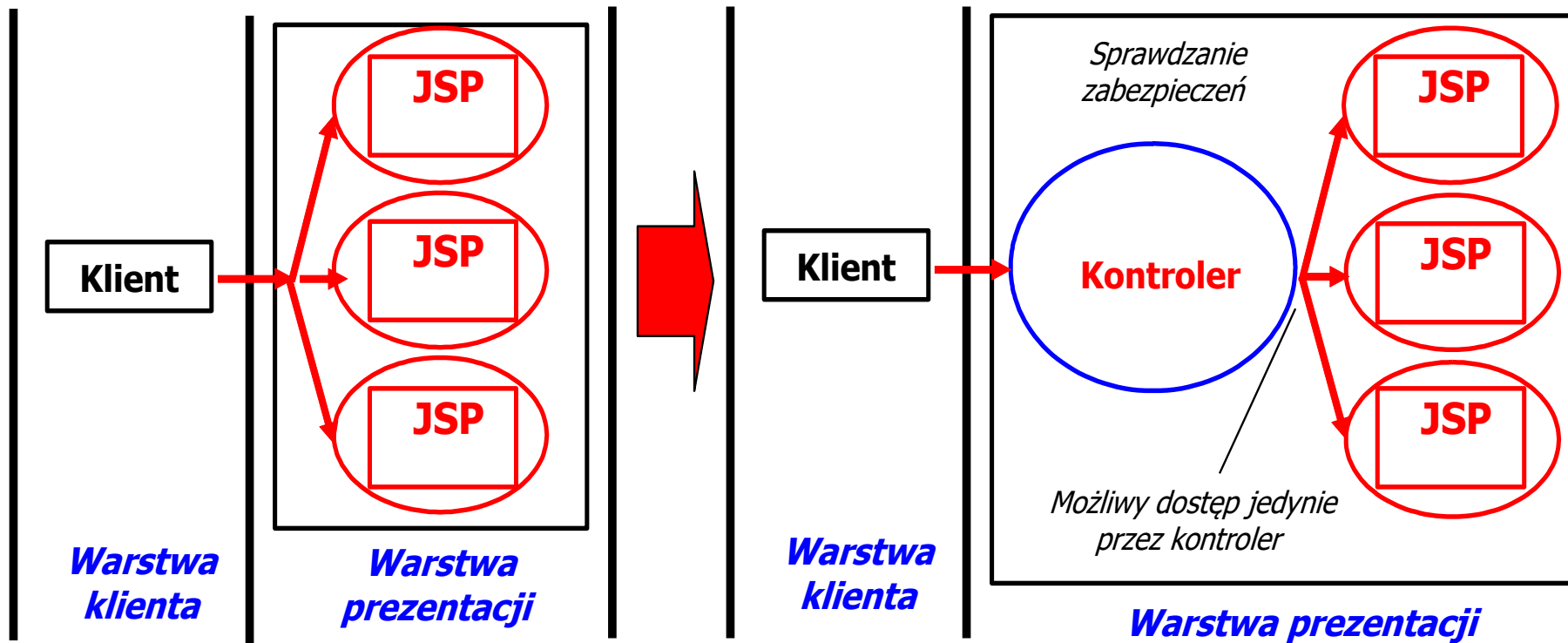
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania części *Warstwy biznesowej* z obiektami typu **POJO**. Wykonanie aplikacji dwuwarstwowej.
6. Przykłady architektury wielowarstwowej aplikacji internetowej typu **EE**. Wykonanie aplikacji typu **EE**. *Warstwa biznesowa*: komponenty typu **EJB** + obiekty **POJO**
7. *Warstwa zasobów (EIS)*- baza danych w systemie baz danych **Derby**
8. Utworzenie obiektowego modelu danych do utrwalania **ORM**
9. *Warstwa integracji*. Zastosowanie wzorca projektowego typu *Domain Store* w technologii **JPA (Java Persistence)** na platformie **Java EE**
10. *Warstwa prezentacji* - **JSF**
11. **Dodatek**

Dodatek

1. Dodanie zabezpieczeń typu Security
2. Refaktoryzacja *Warstwy biznesowej*
3. Tworzenie *Warstwy integracji*

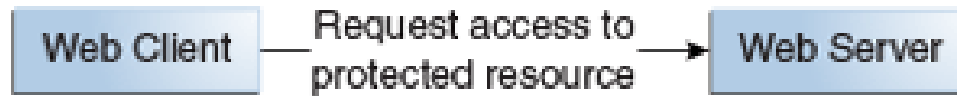
1. Dodanie zabezpieczeń typu **Security**

Ukrywanie zasobów przed klientem za pomocą konfiguracji kontenera – **uwierzytelnianie i autoryzacja**



Przebieg uwierzytelniania i autoryzacji

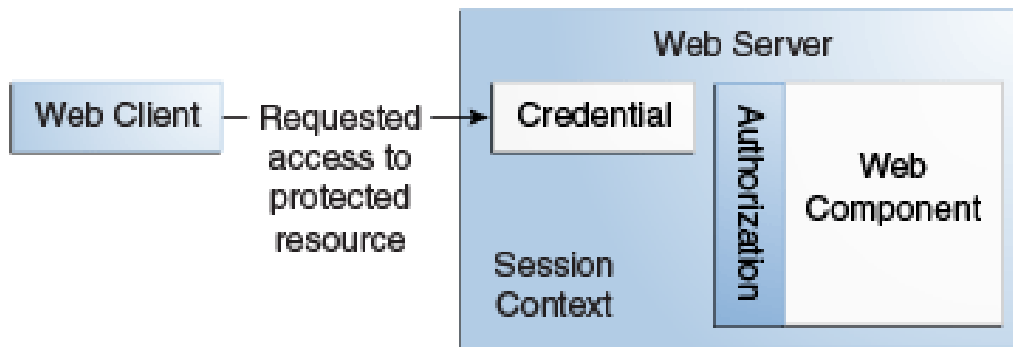
<https://javaee.github.io/tutorial/security-intro001.html>



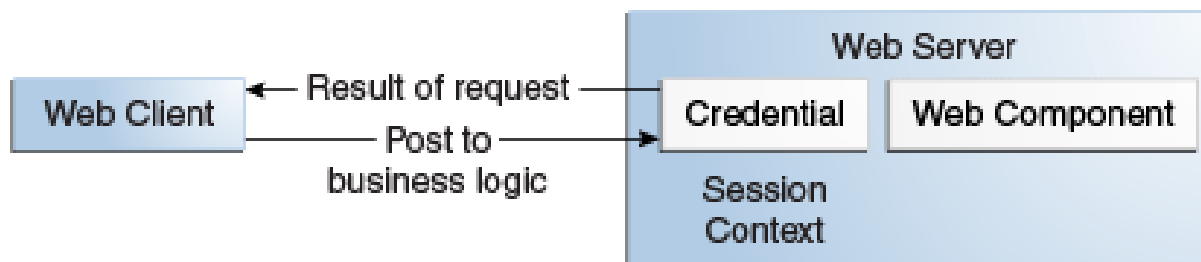
1. Początkowe żądanie do dostępu do chronionych zasobów



2. Pobranie danych do uwierzytelnienia



3. Żądanie dostępu do zasobów oraz kontrola uprawnień tego dostępu (**autoryzacji**)

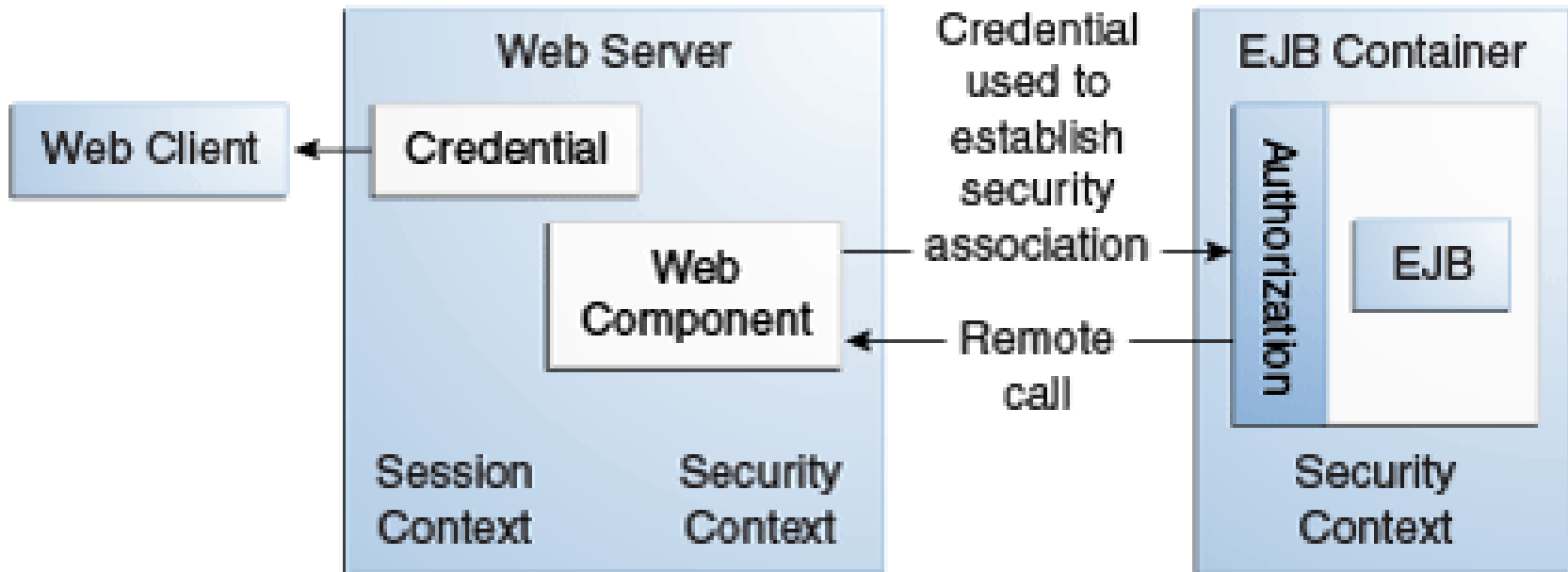


4. Przesłanie wyniku kontroli dostępu. Wynik pozytywny - wysłanie żądania do kontroli dostępu do **Warstwy logiki biznesowej**

Przebieg uwierzytelniania i autoryzacji (cd)

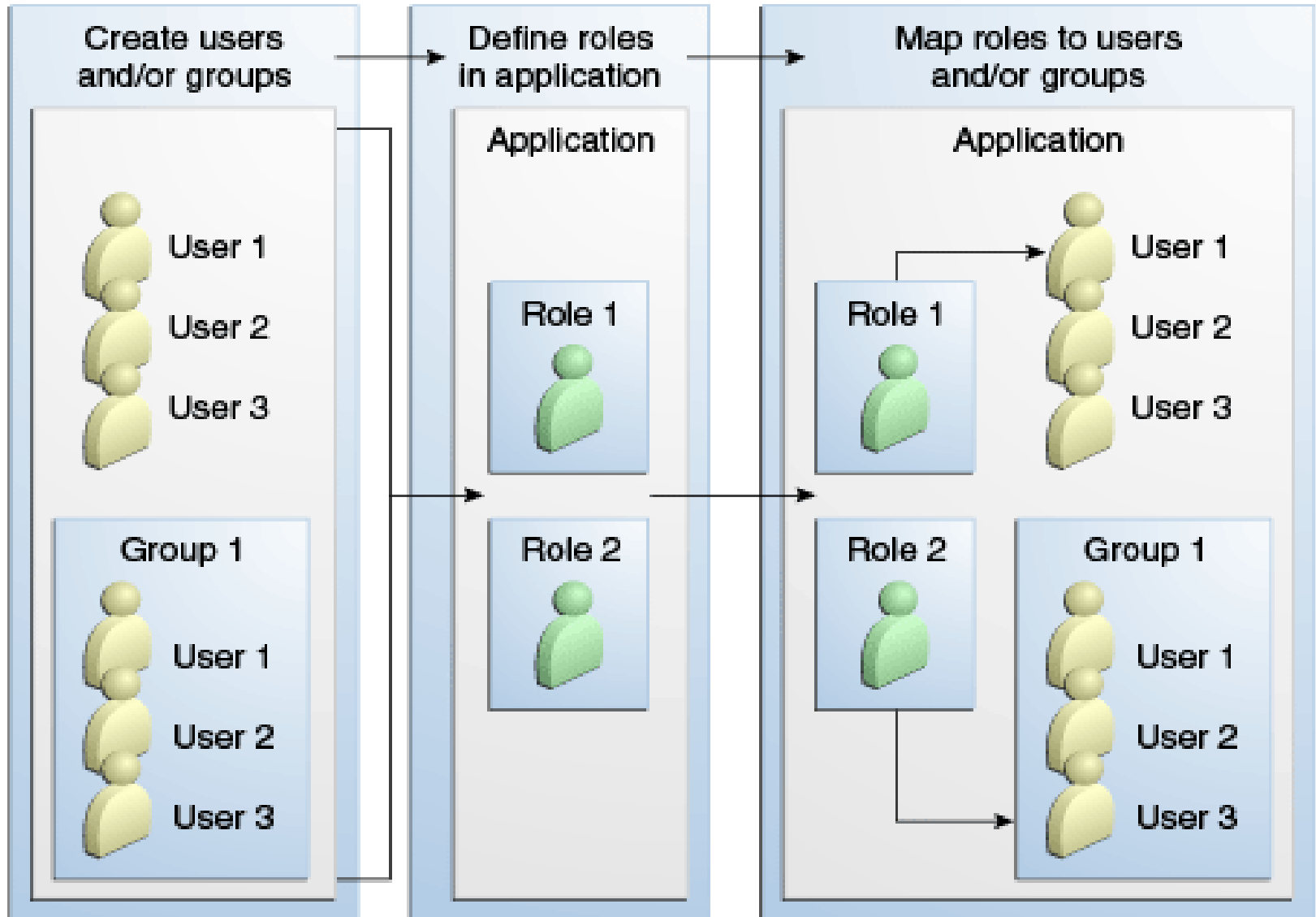
Przebieg autoryzacji w dostępie do *Warstwy logiki biznesowej*

5. Przesłanie wyniku kontroli dostępu z *Warstwy biznesowej* (kontener **EJB**) do *Warstwy prezentacji*. Wynik pozytywny - wysłanie danych do przetwarzania w *Warstwie logiki biznesowej*

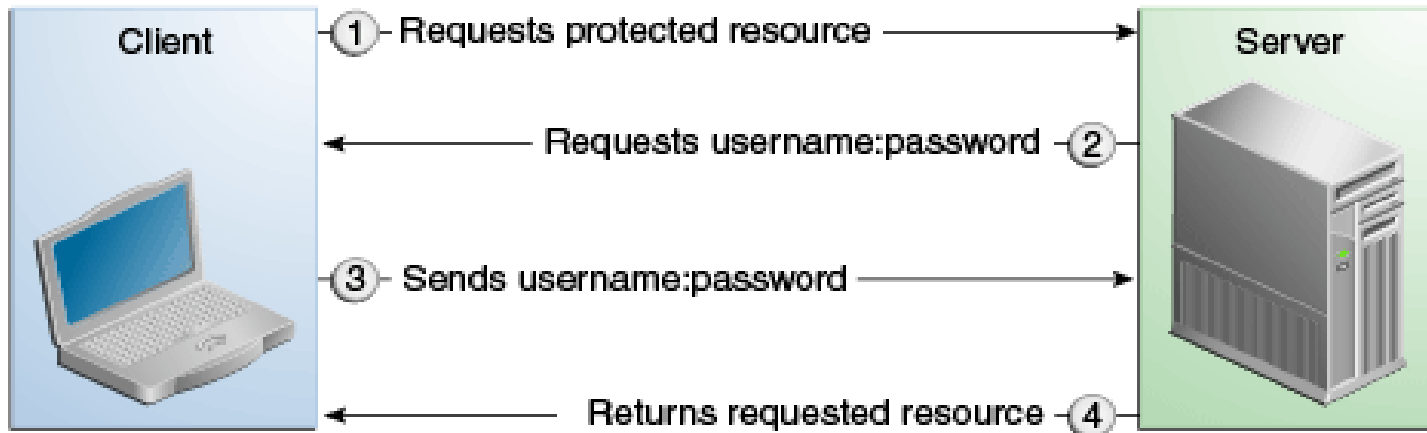


Bazy użytkowników i grup użytkowników. **Role** typu **security** przypisane do **użytkowników i grup**

<https://javaee.github.io/tutorial/security-intro005.html>



Przebieg logowania oparty na nazwie użytkownika i haśle

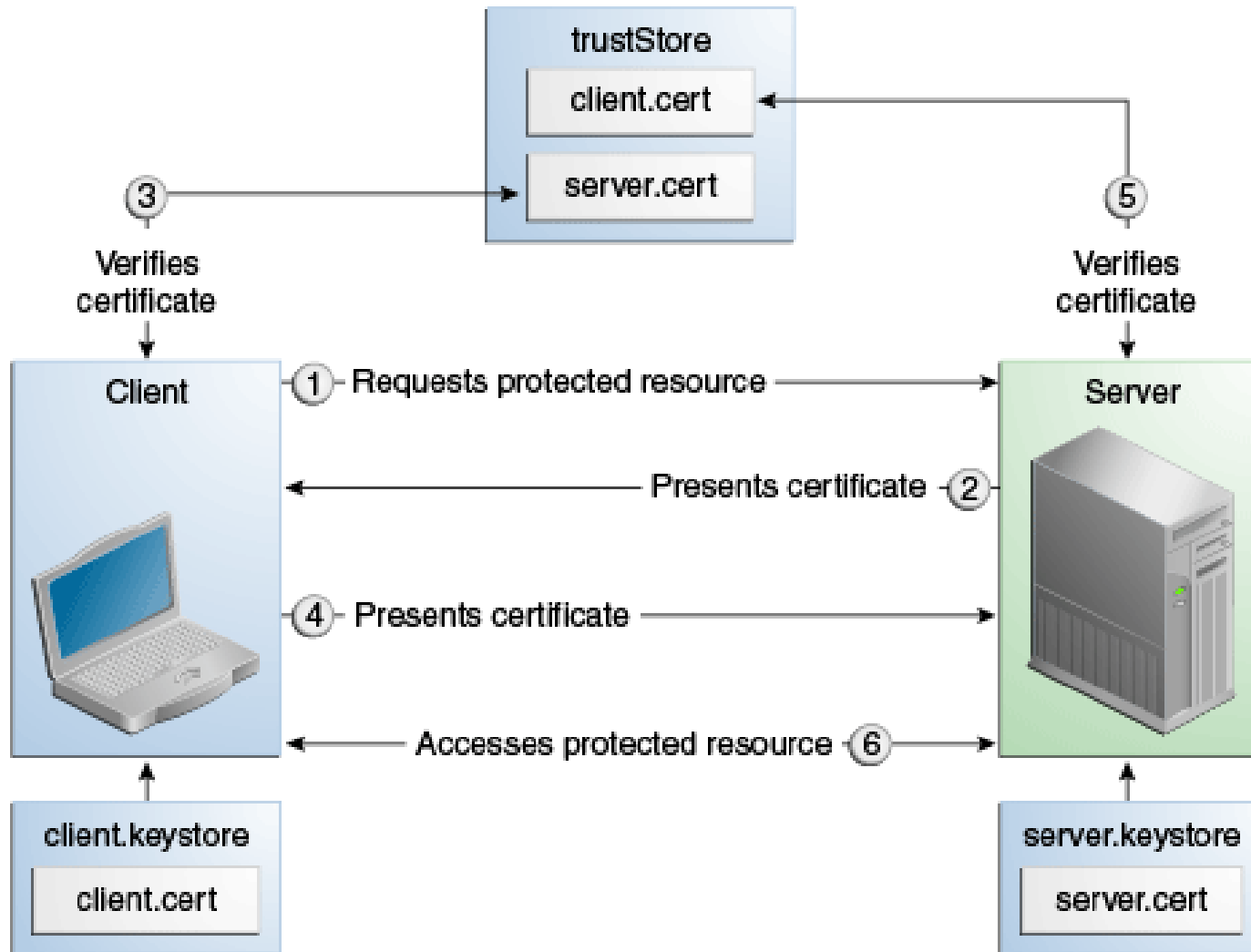


Przebieg logowania oparty na nazwie użytkownika i haśle oraz certyfikacie serwera z wykorzystaniem SSL



Przebieg uwierzytelniania (logowania) z wykorzystaniem certyfikatów klienta i serwera

<https://javaee.github.io/tutorial/security-advanced002.html>



Rodzaje mechanizmów bezpieczeństwa (**security**)

w kontenerach (kod do zarządzania aplikacją przez serwer aplikacji m.in. bezpieczeństwem aplikacji)

- **Deklaratywne mechanizmy bezpieczeństwa**

- deklarowane za pomocą tzw. „*deployment descriptors*” (*web.xml*, *META-INF/ejb-jar.xml*). Deskryptory jako zewnętrzny element aplikacji zawierają informacje specyfikującą role bezpieczeństwa i wymagania dostępu są mapowane w role specyficzne dla środowiska oraz użytkowników i polityki bezpieczeństwa.

[Using Deployment Descriptors for Declarative Security](#)

- **Adnotacje lub metadane** są używane do specyfikowania informacji wewnątrz pliku z kodem klasy.

[Using Annotations to Specify Security Information](#)

Np. `@DeclareRoles("klient")`

```
public class Page1 extends AbstractPageBean { //... }
```

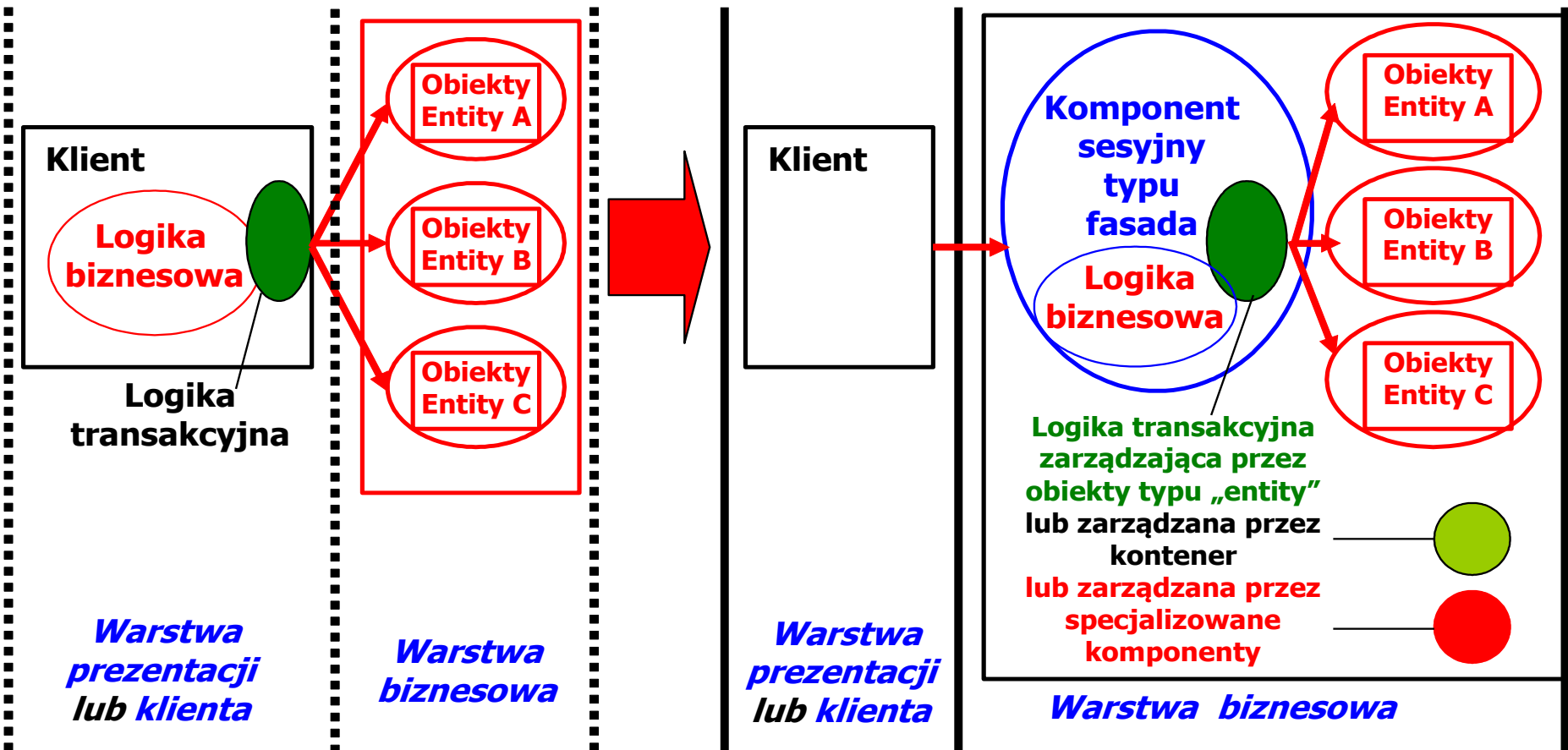
- **Programowe mechanizmy bezpieczeństwa** - są osadzone w aplikacji i służą do podejmowanie decyzji o bezpieczeństwie. Uzupełniają deklaratywne mechanizmy bezpieczeństwa – lepiej wyrażają model bezpieczeństwa aplikacji. API mechanizmów programowych: [Using Programmatic Security](#)

- zabezpieczenia komponentów typu EJB - metody interfejsu **EJBContext**
- zabezpieczanie aplikacji internetowych - metody interfejsów **SecurityContext**, **HttpServletRequest**. Metody te pozwalają na podejmowanie decyzji biznesowych opartych na rolach bezpieczeństwa nadawcy lub zdalnego odbiorcy

2. Refaktoryzacja warstwy biznesowej (podsumowanie)

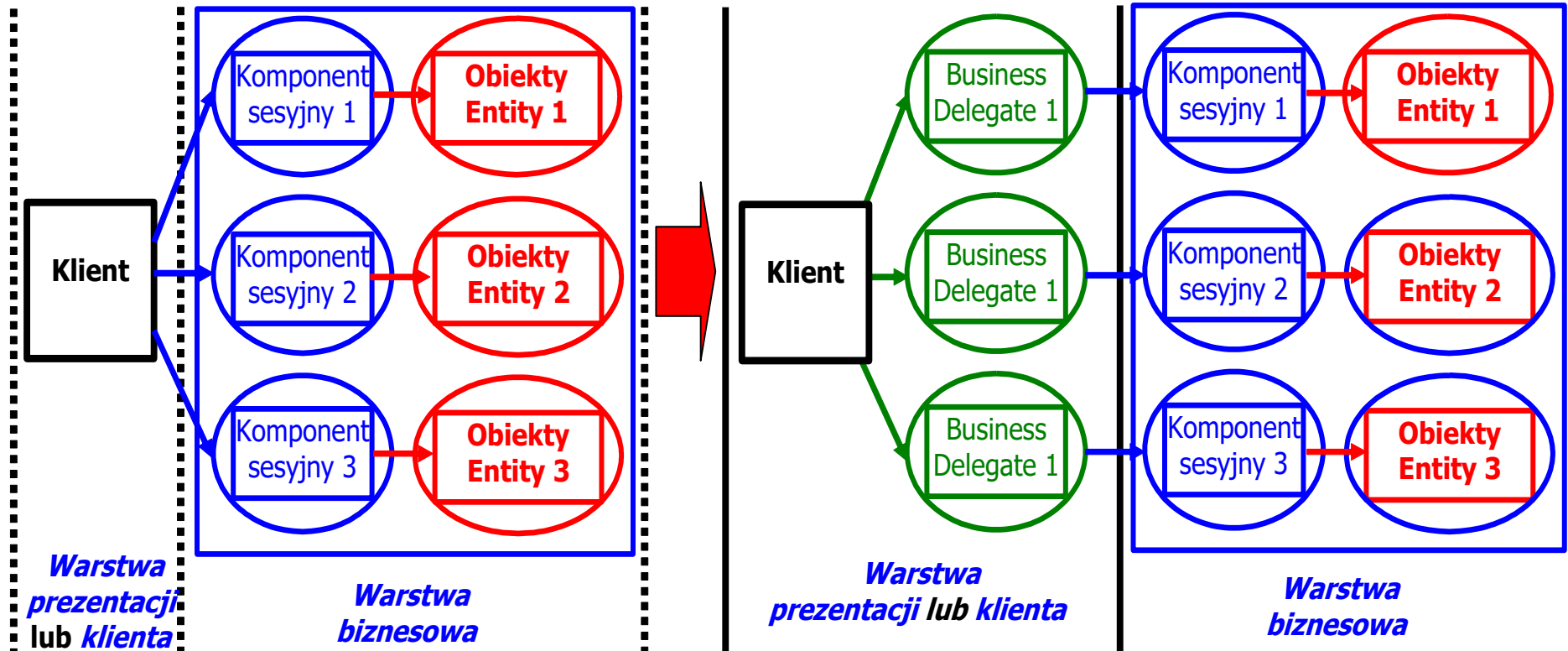
Refaktoryzacja warstwy biznesowej 1

Obiekty danych typu **Entity** (obiekty biznesowe) z **Warstwy biznesowej** są udostępniane klientom w innych warstwach za pomocą **fasadowych komponentów sesyjnych typu Control** (komponent typu fasada - hermetyzujący dostęp do usług biznesowych)



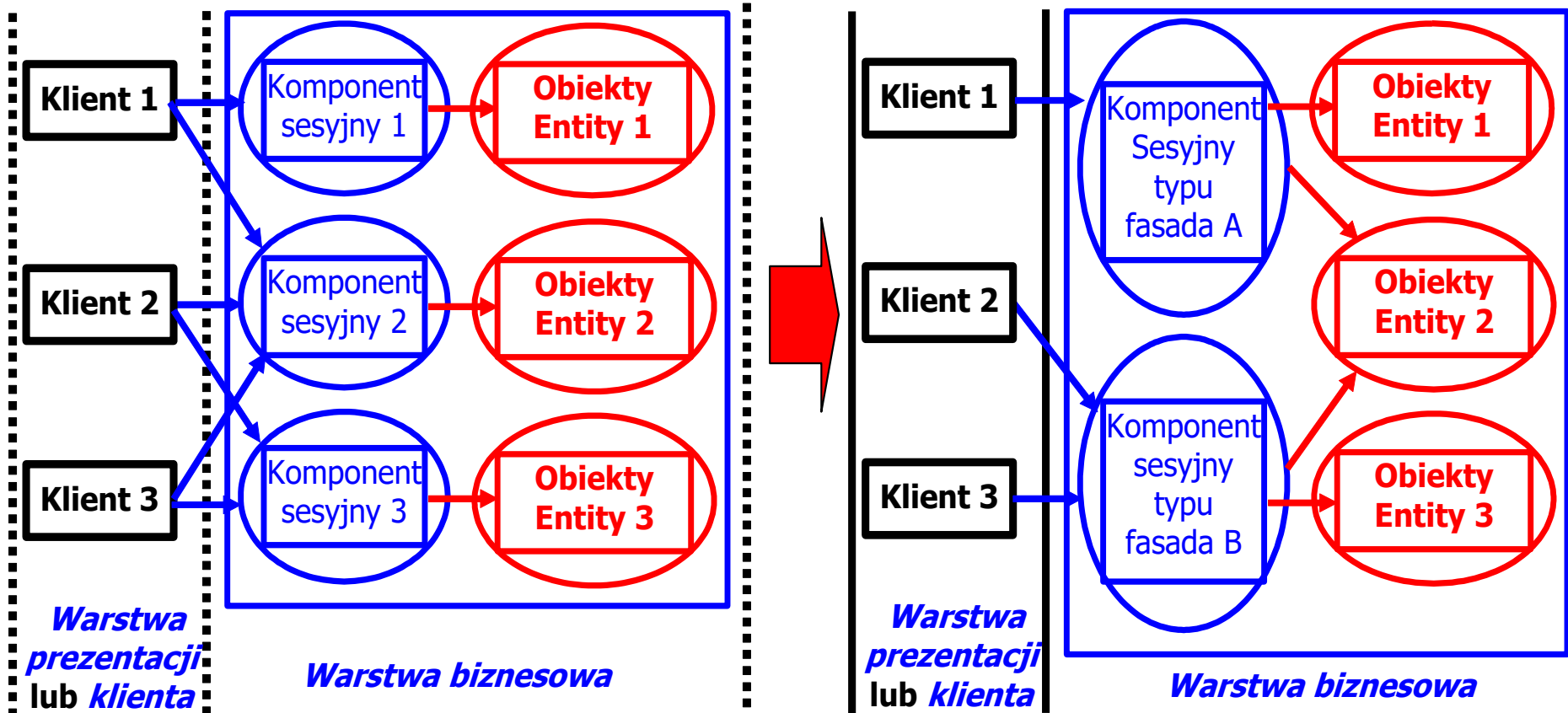
Refaktoryzacja *Warstwy biznesowej 2*

Komponenty sesyjne typu **Control** (pośredniczące w dostępie do **obiektów danych typu Entity**) z *Warstwy biznesowej* są udostępniane klientom w innych warstwach za pomocą **obiektów fasadowych typu Control** (**hermetyzujących dostęp do Warstwy biznesowej** - czyli komponentów **Business Delegate**)



Refaktoryzacja *Warstwy biznesowej* 3

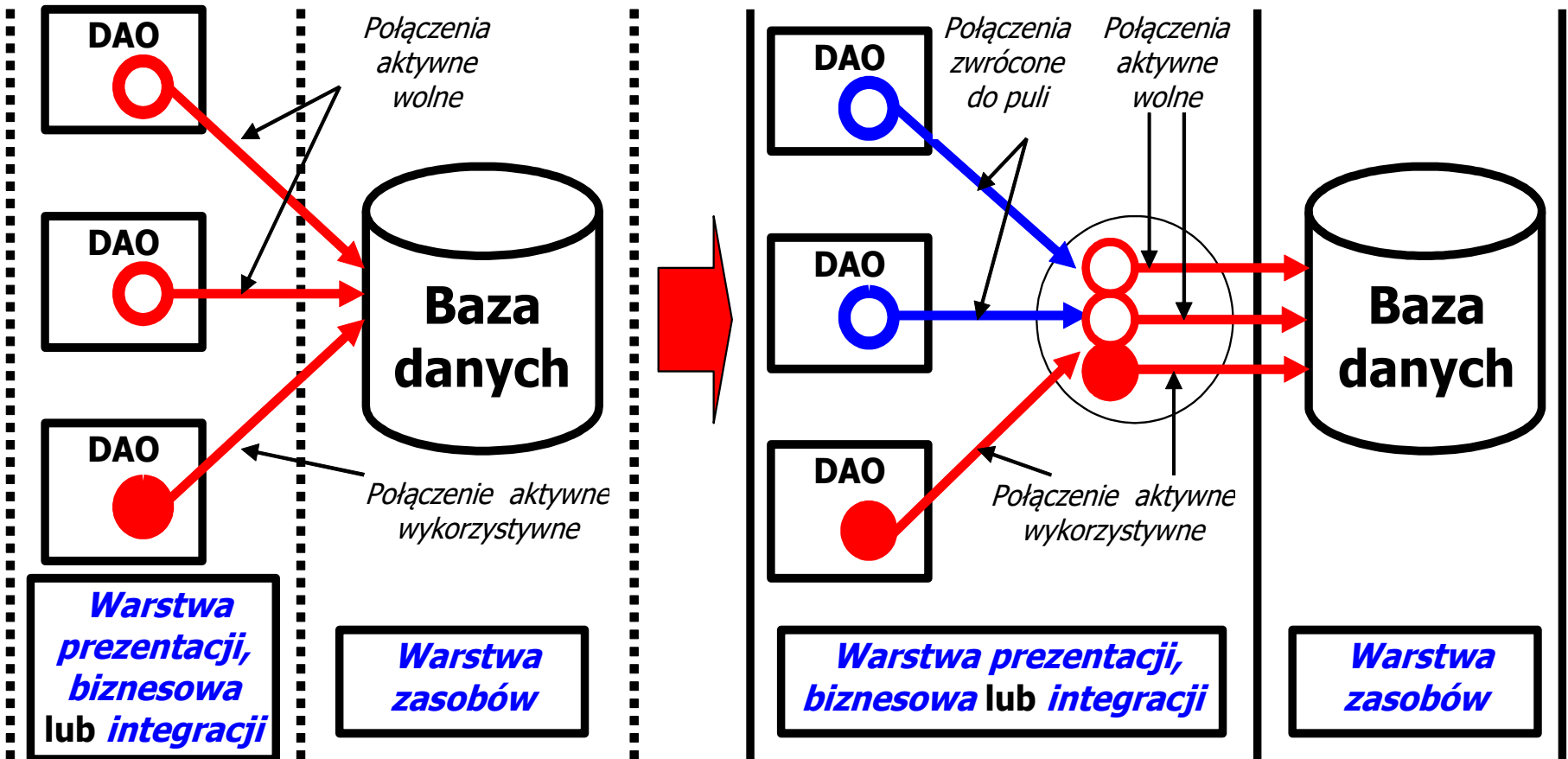
Sesyjne komponenty fasadowe typu **Control** (każdy komponent jako **odrębna usługa biznesowa**), hermetyzujące **obiekty danych typu Entity** z *Warstwy biznesowej* są udostępniane klientom w innych warstwach. Zwykle obiekty sesyjne są jedynie pośrednikami obiektów **Entity**, natomiast hermetyzują całe usługi, które wymagają odwołania do wielu zwykłych komponentów sesyjnych.



3. Tworzenie *Warstwy integracji* (podsumowanie)

Refaktoryzacja dostępu do danych – pula połączeń

- Liczba połączeń kodu dostępu do danych (**DAO**) z bazą danych jest ograniczona
- Połączenia kodu dostępu do danych (**DAO**) nie zawsze są wykorzystywane, lecz są utrzymywane, ponieważ otwarcie połączenia z bazą danych zabiera czas i zasoby
- **Pula połączeń** kodu dostępu do danych (**DAO**) pozwala racjonalnie zarządzać połączeniami aplikacji z bazą danych



Wydzielanie kodu dostępu do danych

- Kod dostępu do danych jest wydzielany z klas, które są używane do spełniania również innych celów
- Kod dostępu do danych powinno umieszczać się logicznie i fizycznie bliżej źródła danych

