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

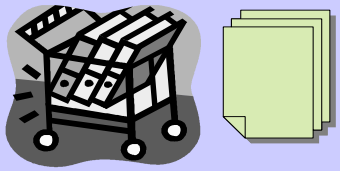
1. **Wielowarstwowa architektura systemu informatycznego**

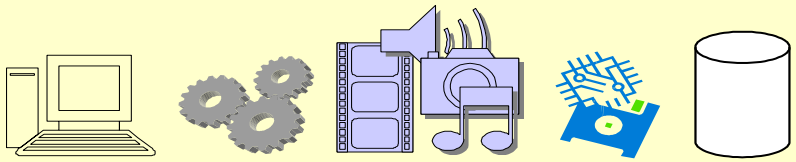
Definicja systemu informatycznego (wykład 1)

(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

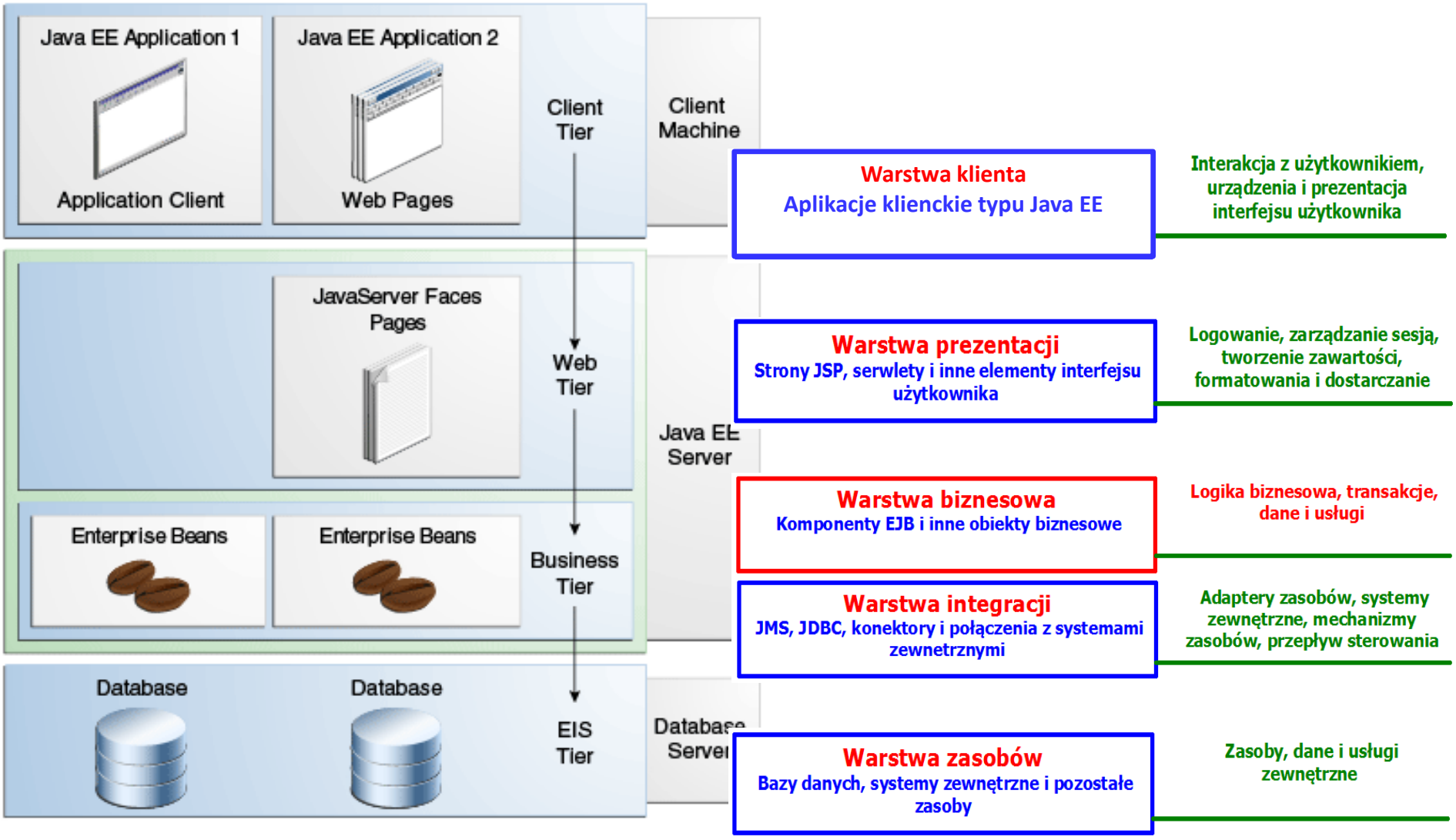
System informatyczny
jest to zbiór powiązanych ze sobą elementów nieformalnych, formalnych i technicznych, którego funkcją jest przetwarzanie danych przy użyciu techniki komputerowej

Techniczny system informacyjny

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

Java EE 8: Distributed Multitiered Applications (javaee.github.io)

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

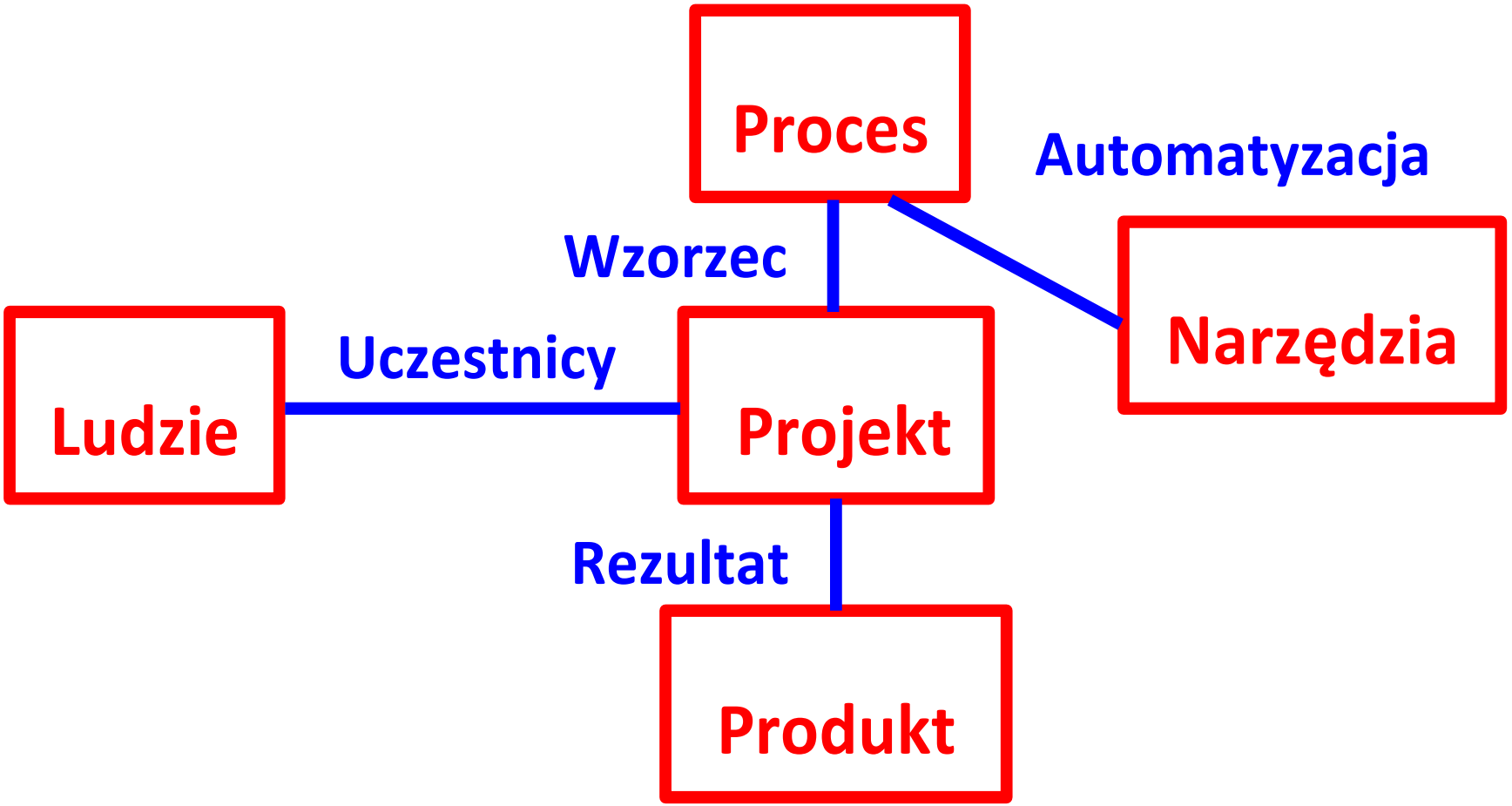


Platformy Javy - slajd 3 (**Java SE**, **Java EE**, Java ME, Java Card)

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**
  - **Wielożywalności**
  - **Zrozumiałości**
  - **Stopnia osiągniętej abstrakcji**

Przykłady powiązania metryk kodu z oceną oprogramowania

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

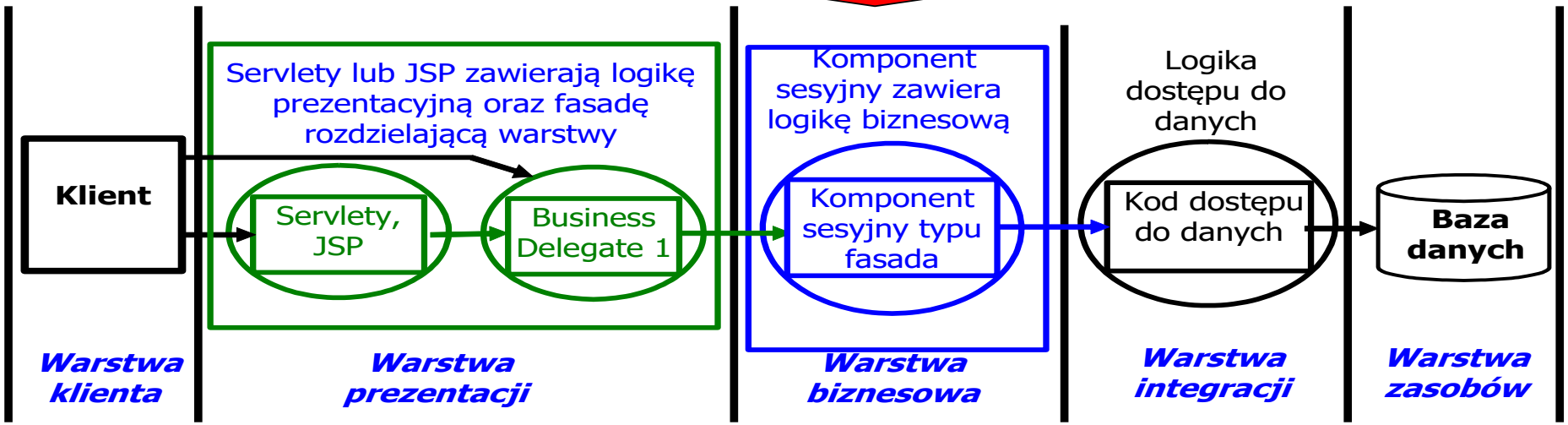
Definicje metryk - Definicje metryk oprogramowania

Ocena jakości - Zarządzanie jakością oprogramowania

Narzędzia - Java Code Quality Tools Recommended by Developers - DZone

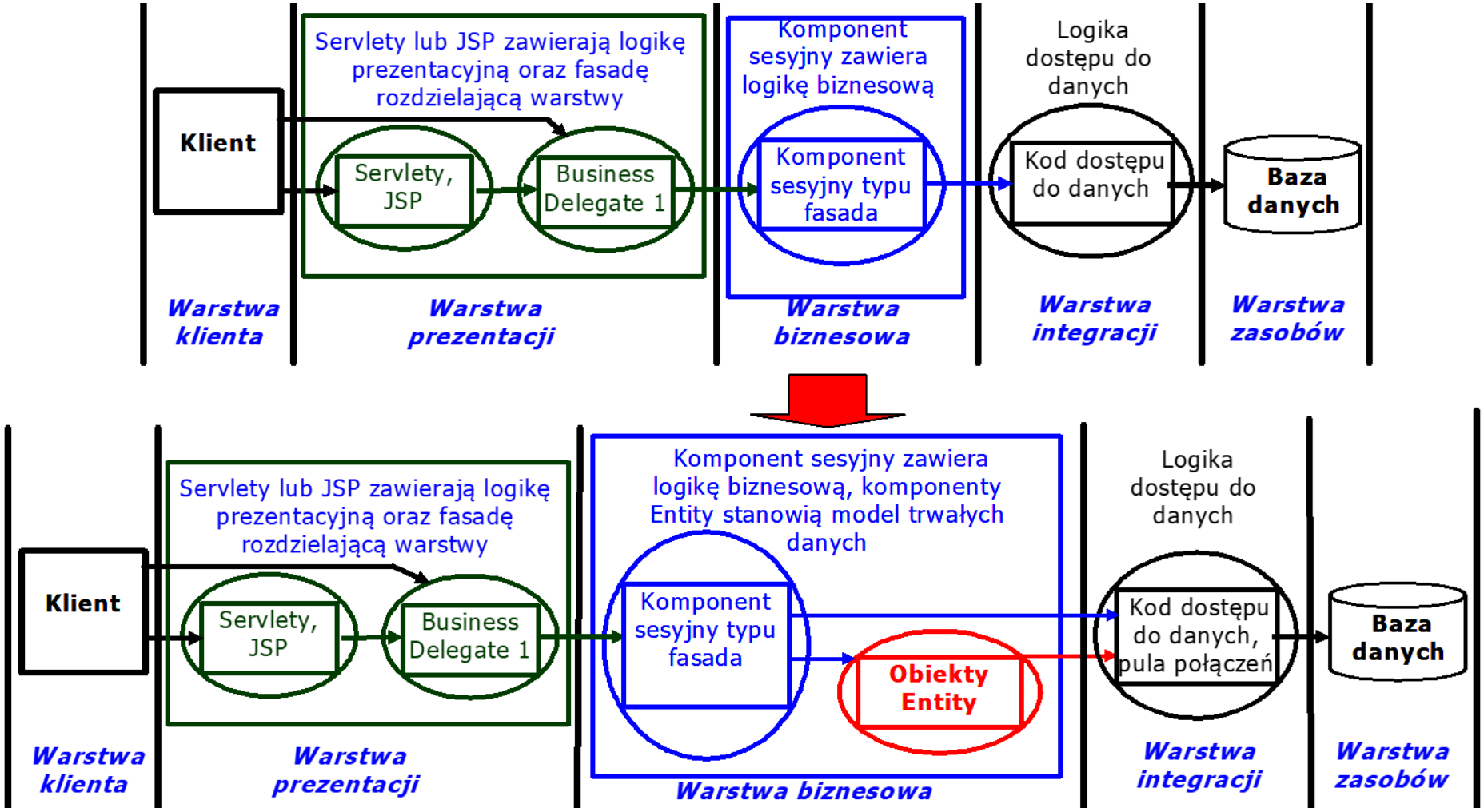
Refaktoryzacja architektury wielowarstwowej - część 1

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

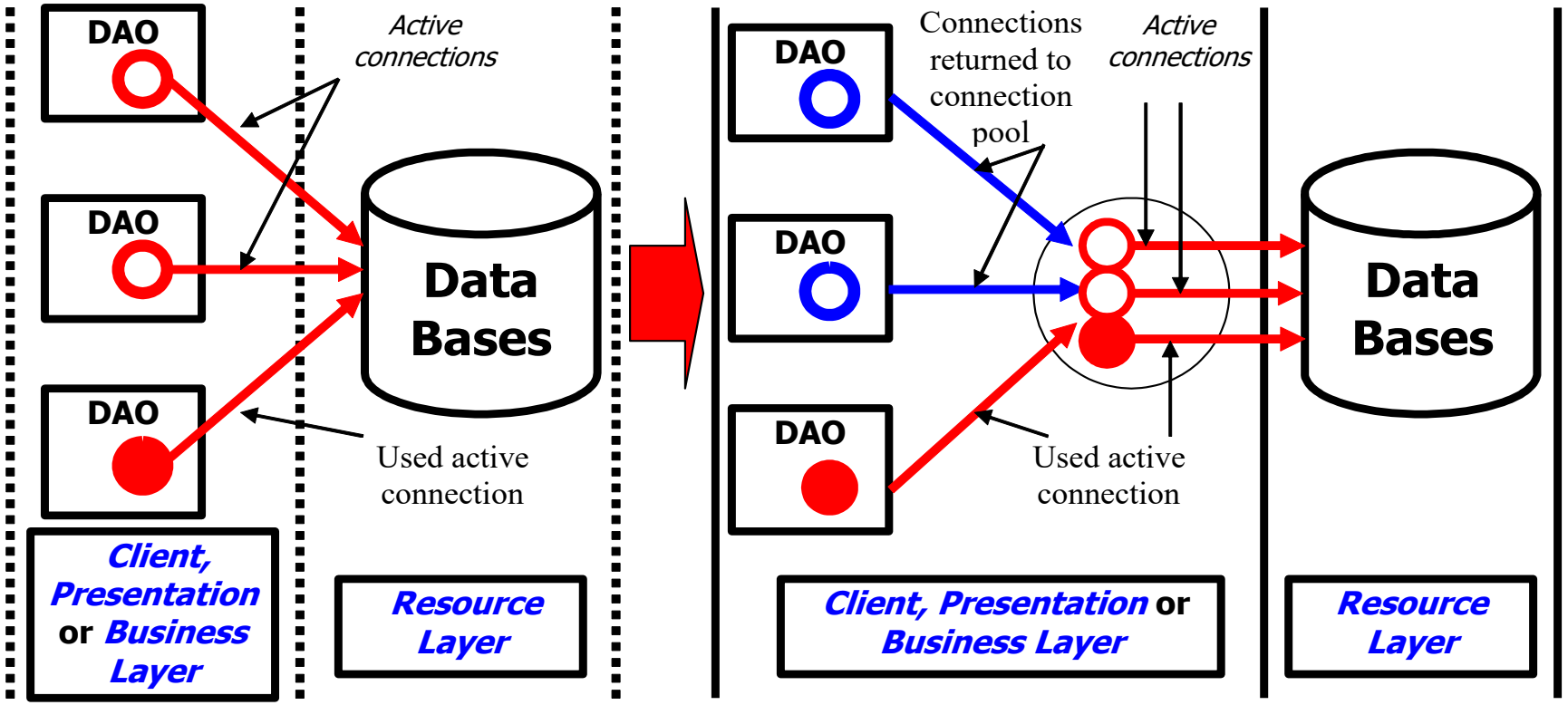
3. Należy zdefiniować obiektowy model danych, który zbudowany jest z obiektów danych typu **Entity**



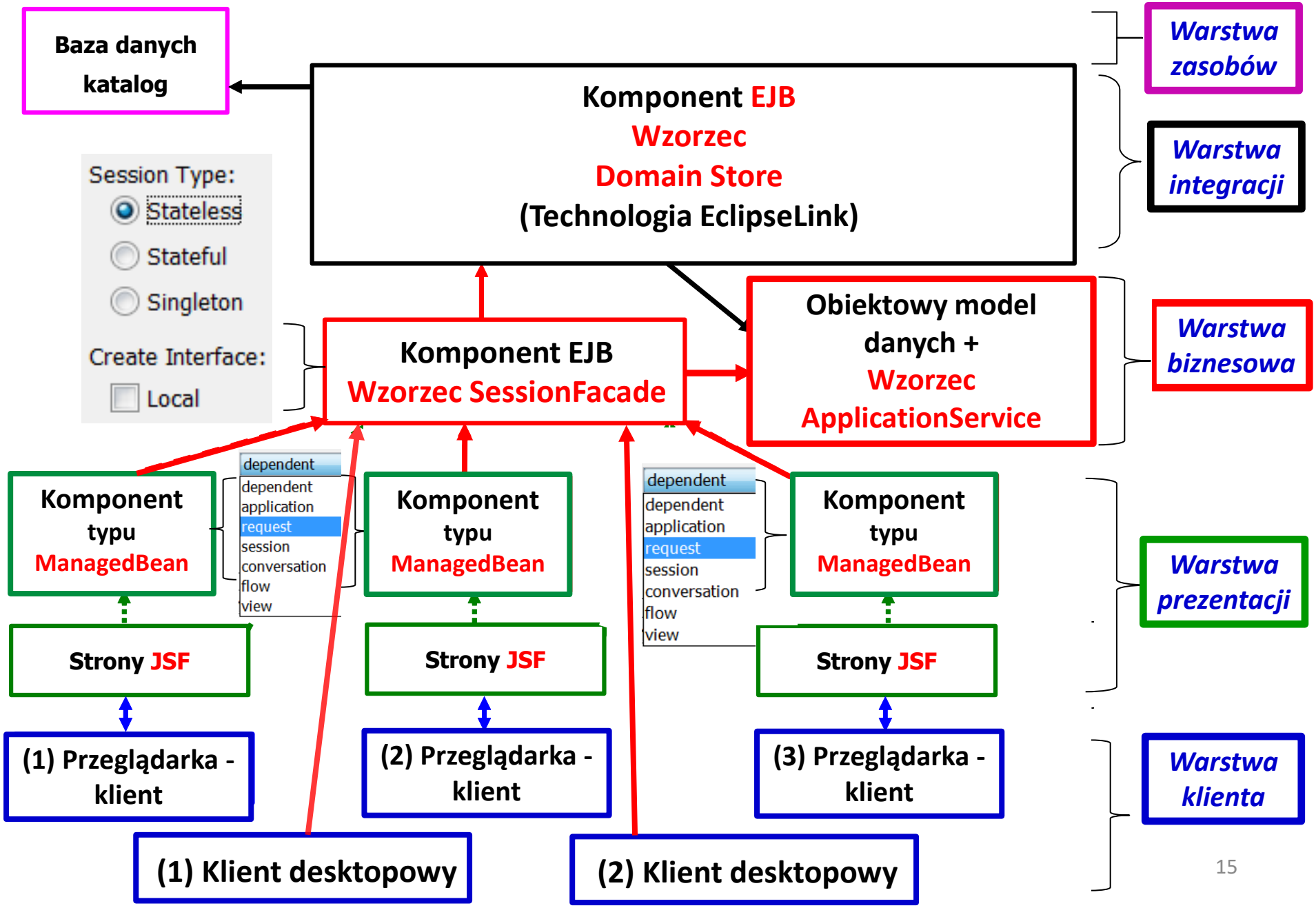
Zarządzanie połączeniami do bazy danych – pula połączeń

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

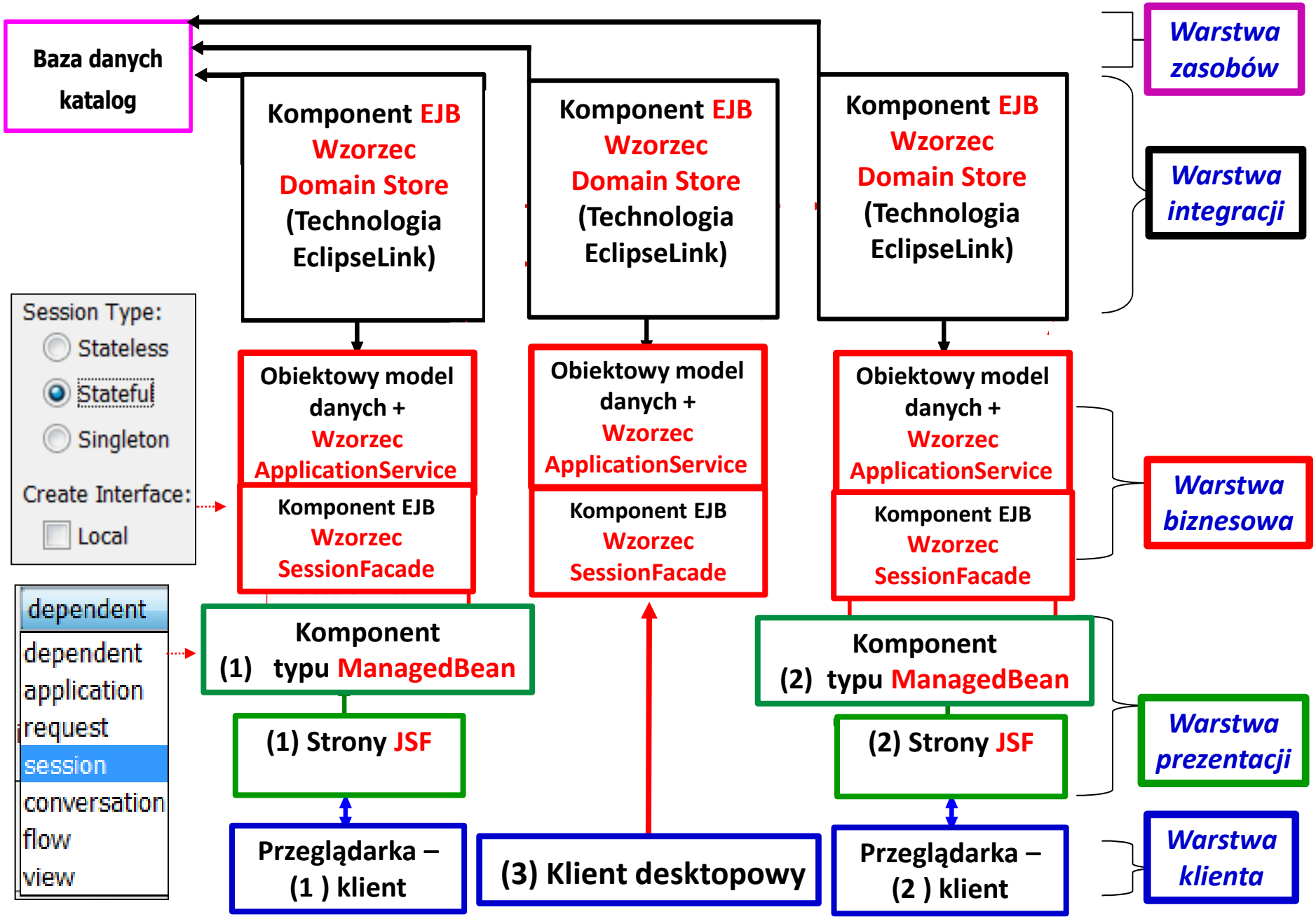
Pula wspólnych połączeń poprawia wydajność i skalowalność aplikacji



Architektura aplikacji pięciowarstwowej -Java EE 8.0 JavaServer Faces



Architektura aplikacji pięciowarstwowej – Java EE 8.0 JavaServer Faces



Zagadnienia

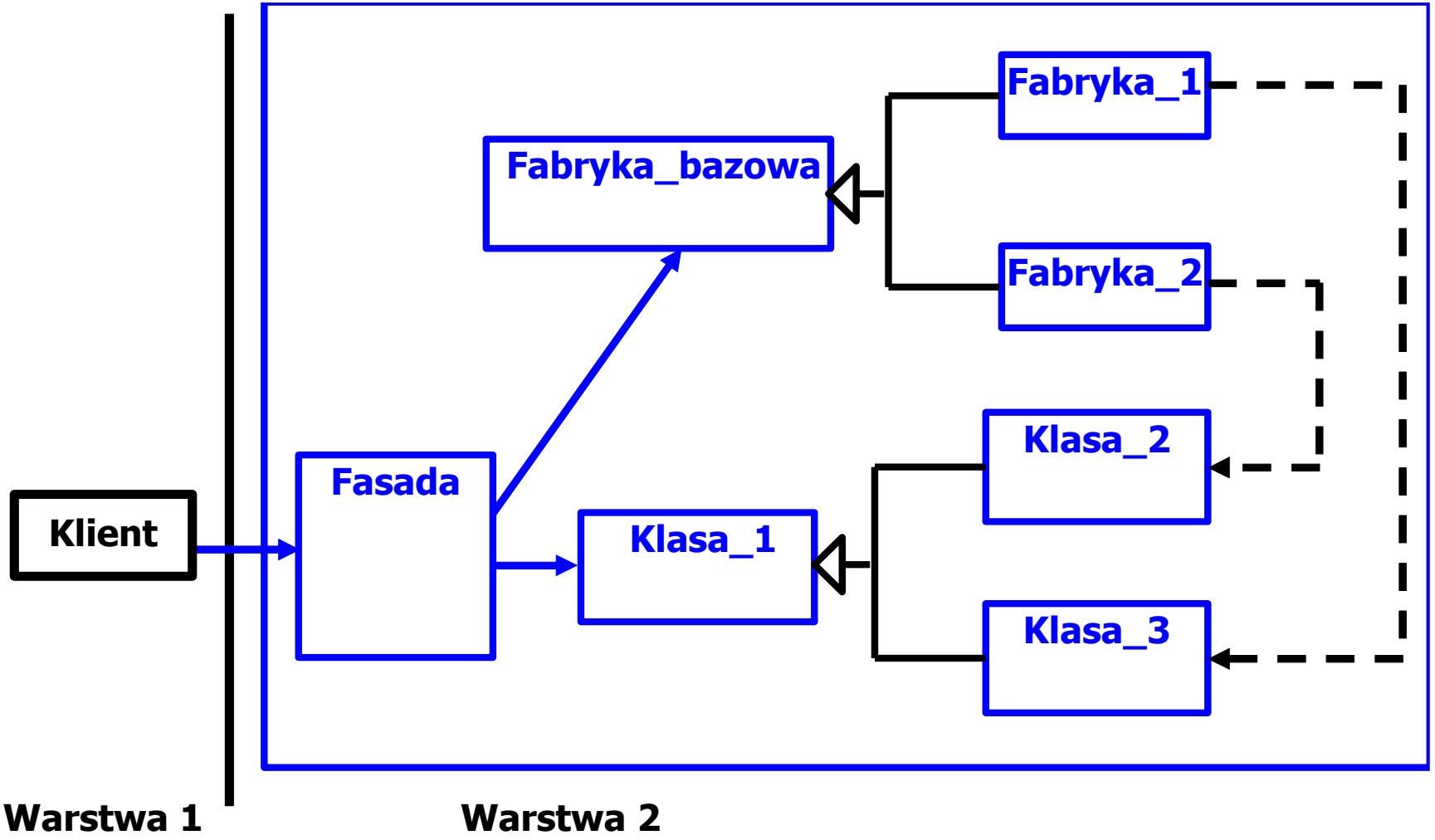
1. **Wielowarstwowa architektura systemu informatycznego**
2. **Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury**
3. **Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej [7]**

Identyfikacja wzorców projektowych (wykład 5 – część 2)

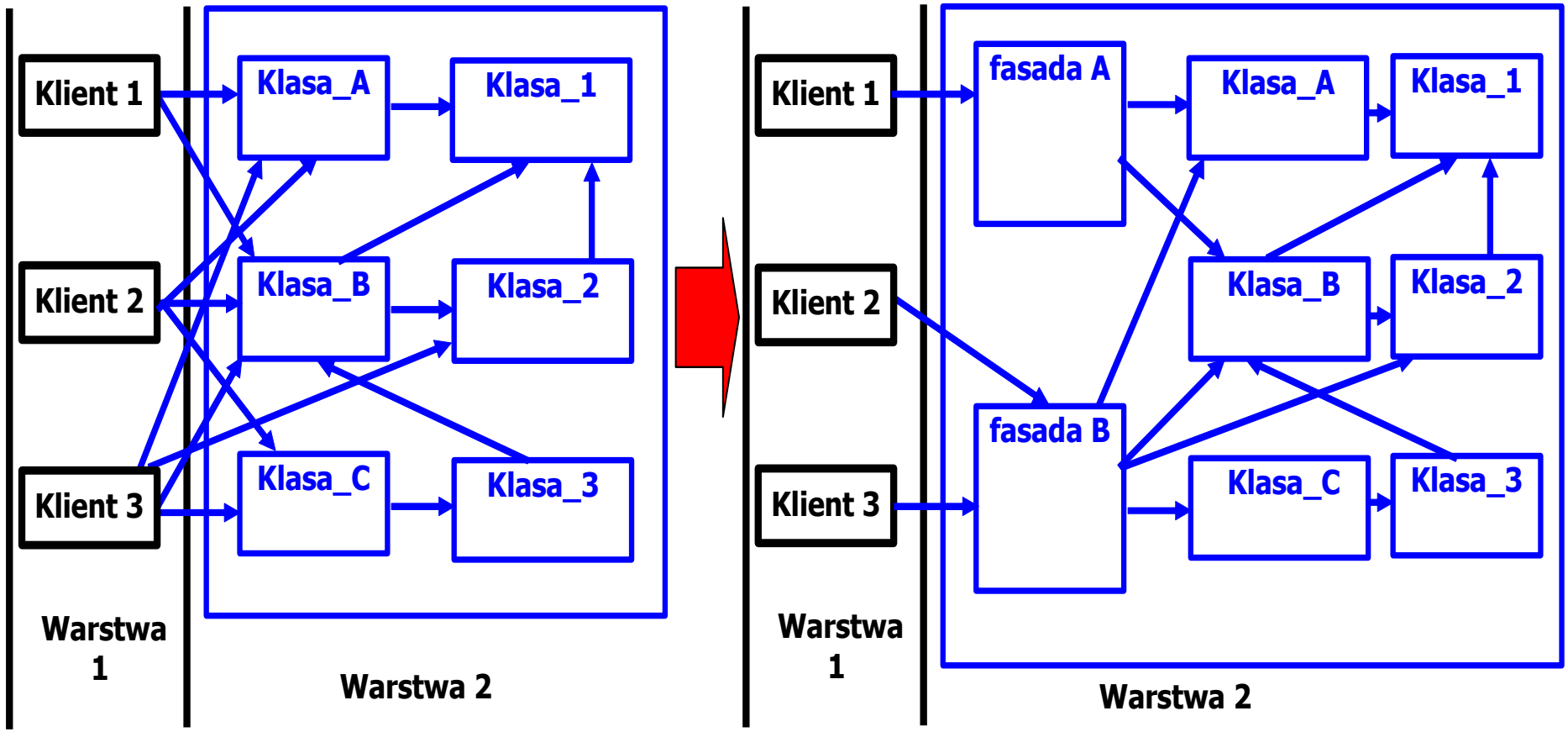
- Dobrze zbudowany system obiektowy jest pełen wzorców obiektowych
- Wzorzec to zwyczajowo przyjęte rozwiązanie typowego problemu w danym kontekście
- Strukturę wzorca przedstawia się w postaci diagramu klas
- Zachowanie się wzorca przedstawia się za pomocą diagramu sekwencji
- **Wzorce projektowe: Wzorzec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

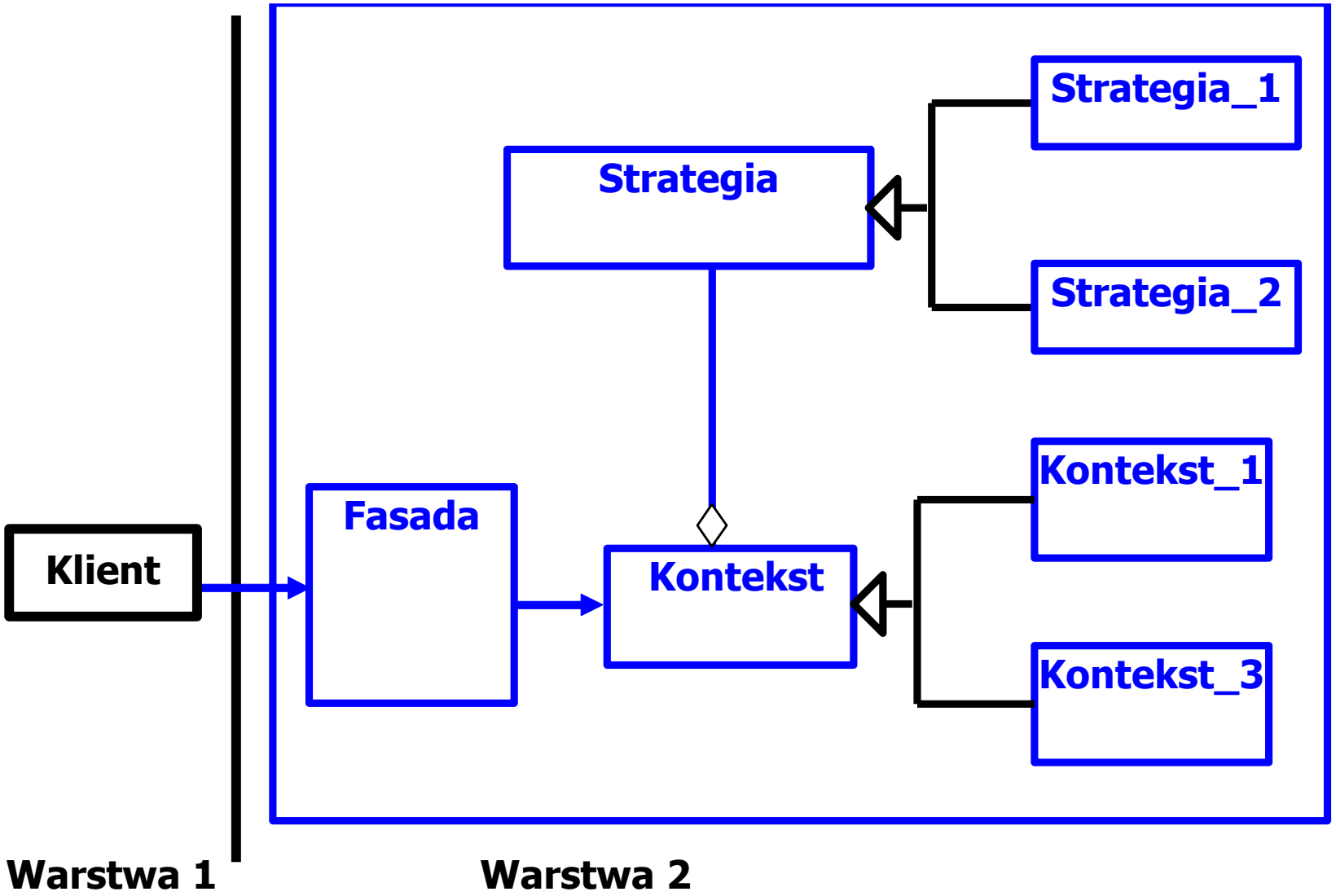
3.1. Wzorzec uniwersalny kreacyjny stosowany w każdej z warstw: *Fabryka obiektów* (wykład 5 – część 2) – oddzielenie tworzenia obiektów od zarządzania nimi i używania ich



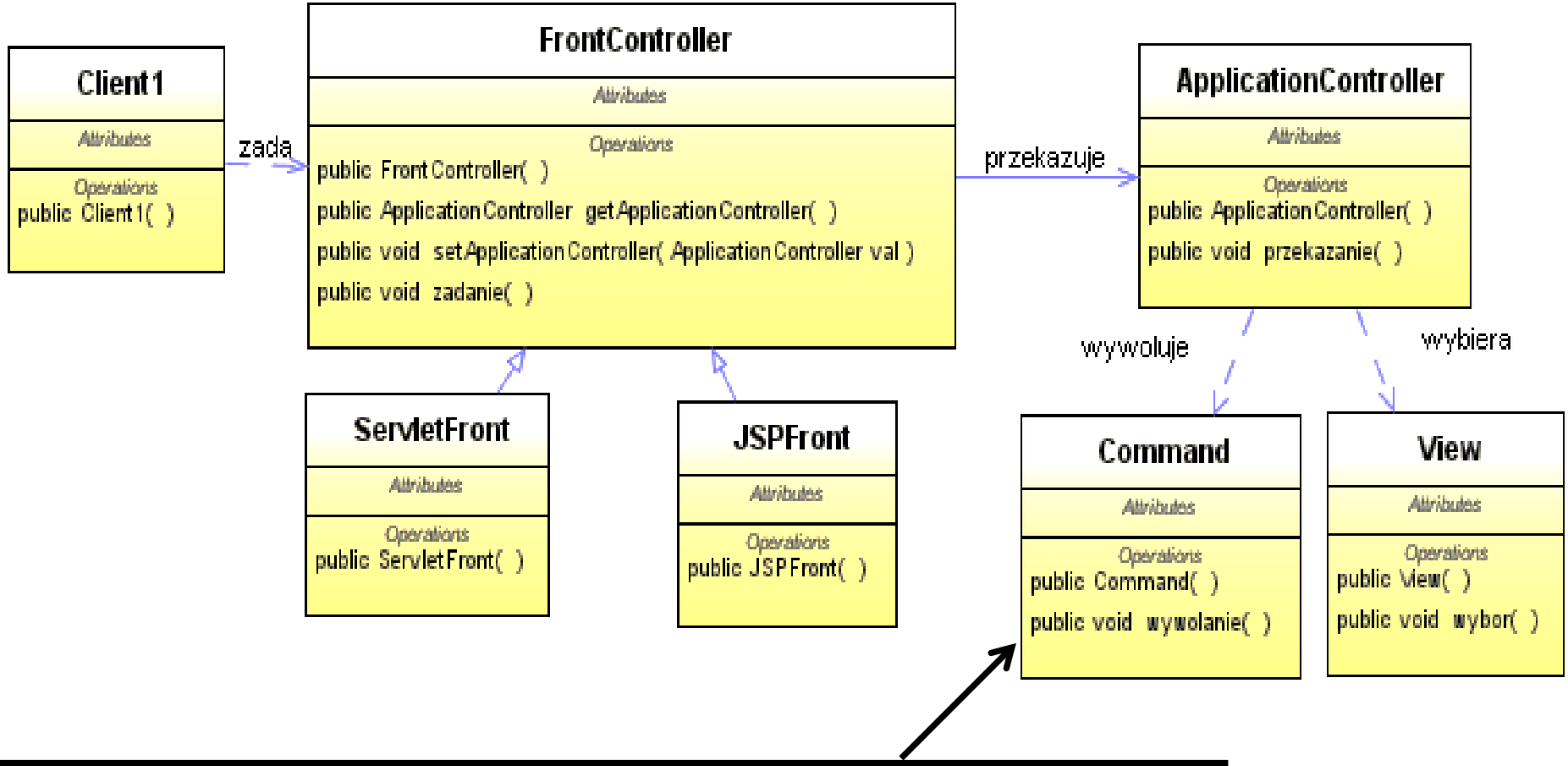
3.2. Wzorzec uniwersalny strukturalny: *Fasada* (wykład 5 – część 2) – hermetyzacja logiki biznesowej



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

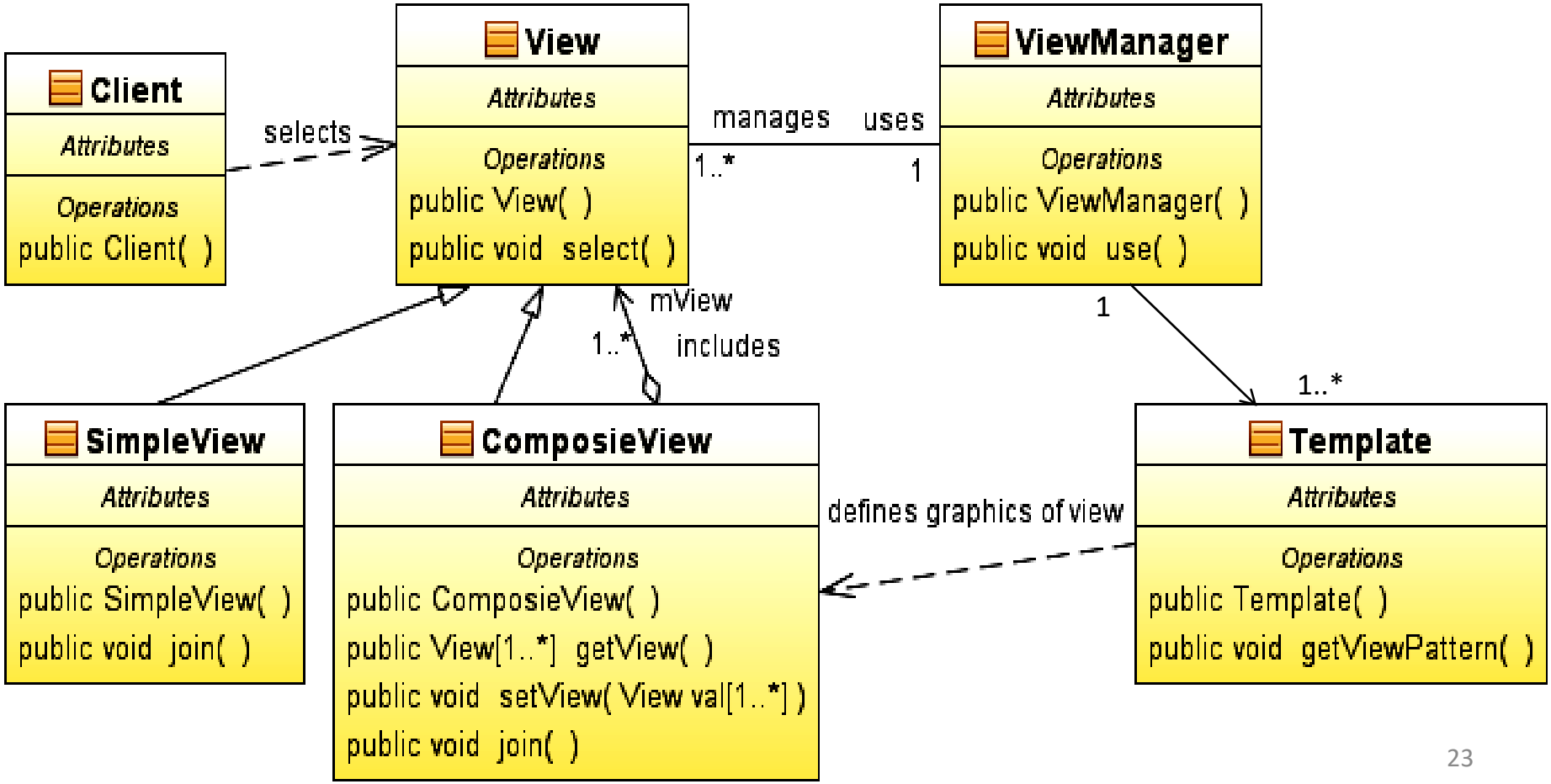


3.4. Wzorzec EE *Warstwy prezentacji*: *FrontController* – scentralizowany punkt dostępowy do obsługi żądań w *Warstwie prezentacji*

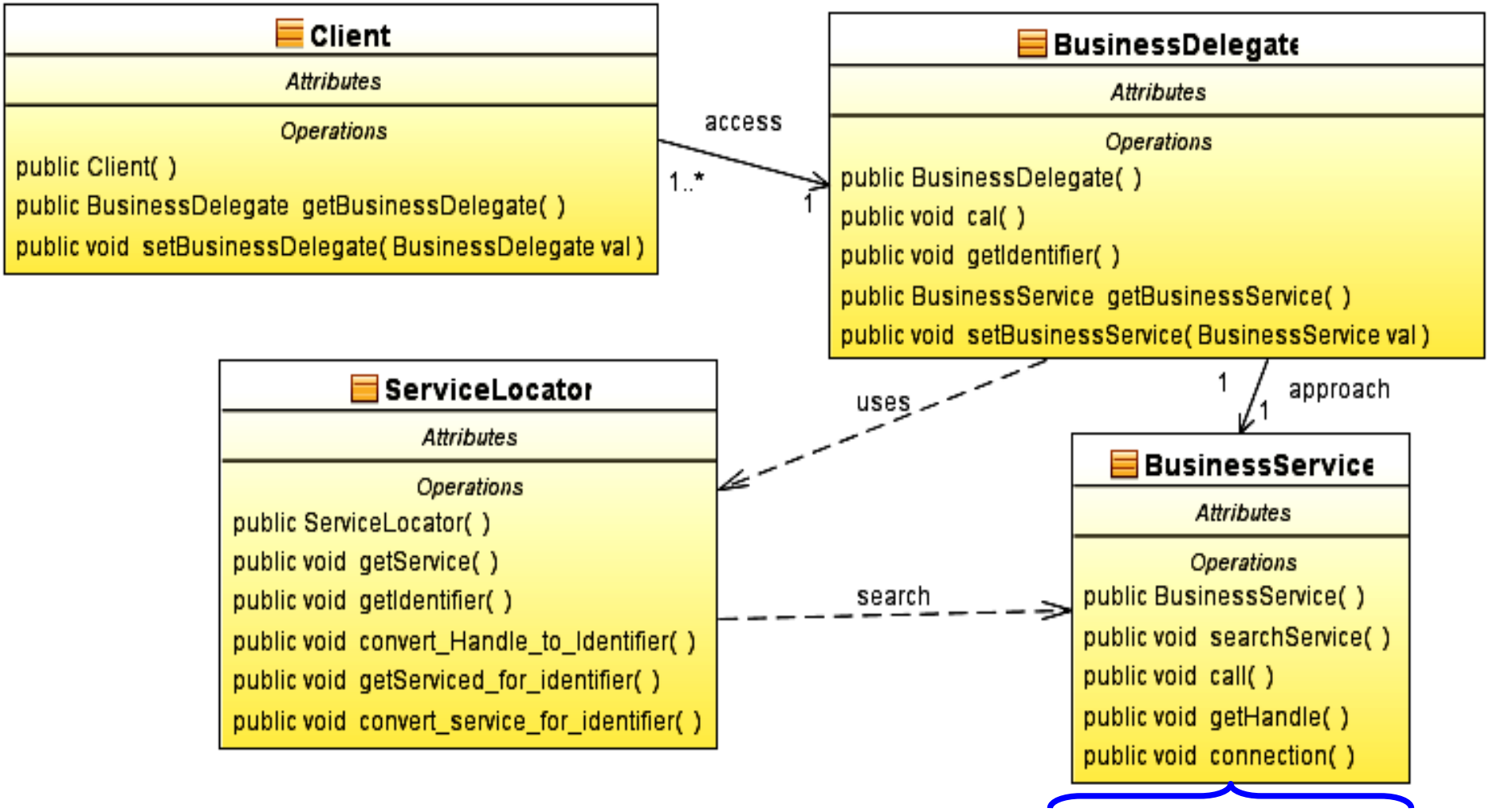


Wywołanie metod z *Warstwy biznesowej* np. za pośrednictwem wzorca **ApplicationService** (wzorzec *Warstwy biznesowej*)

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*).

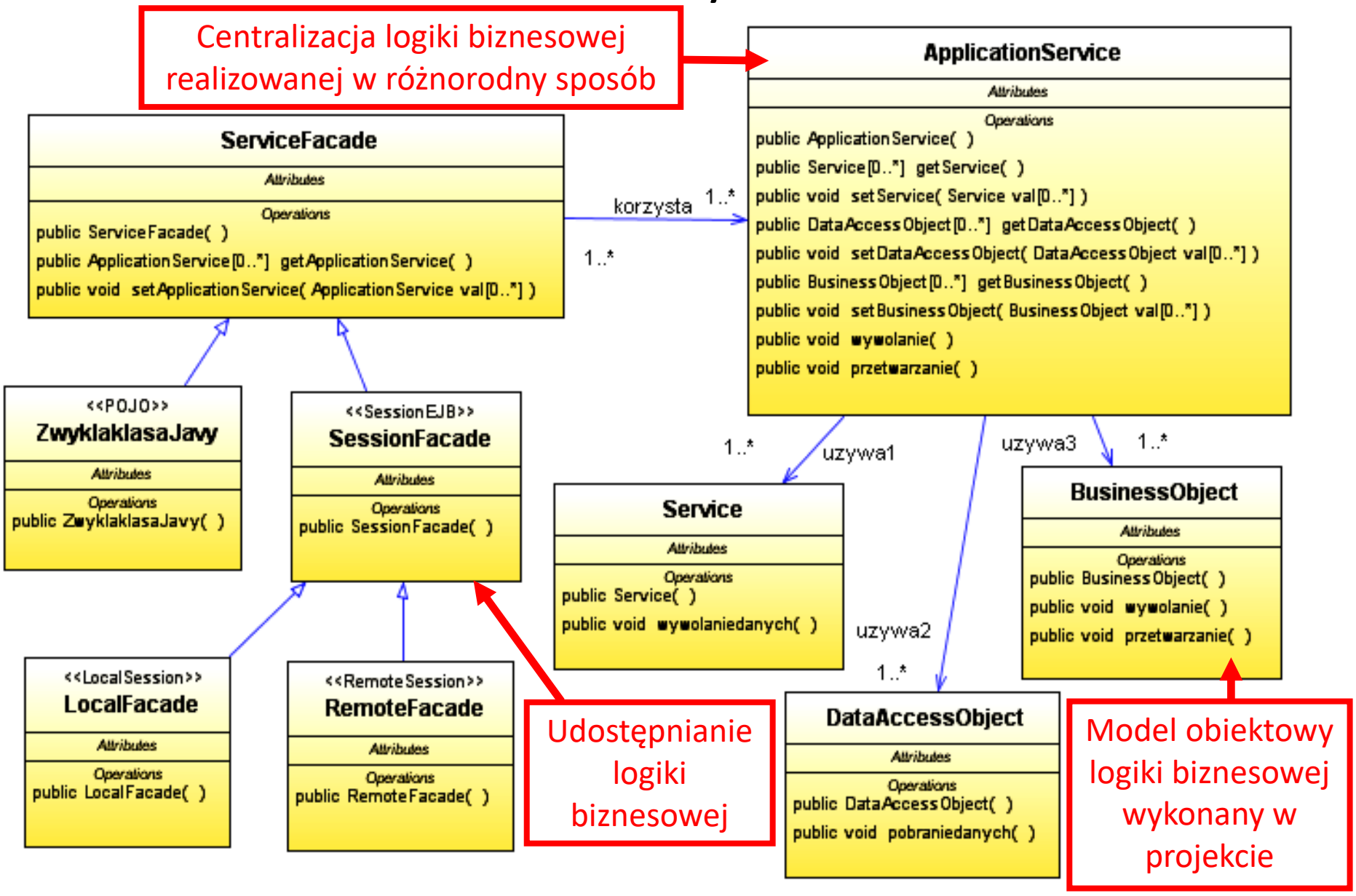


3.6. Wzorzec EE *Warstwy prezentacji*: do zdalnego wywołania usług z *Warstwy klienta* w celu ukrycia złożoności zdalnej komunikacji z komponentem usług biznesowych - *BusinessDelegate* (*Client==* wzorzec *Warstwy prezentacji ApplicationController*)

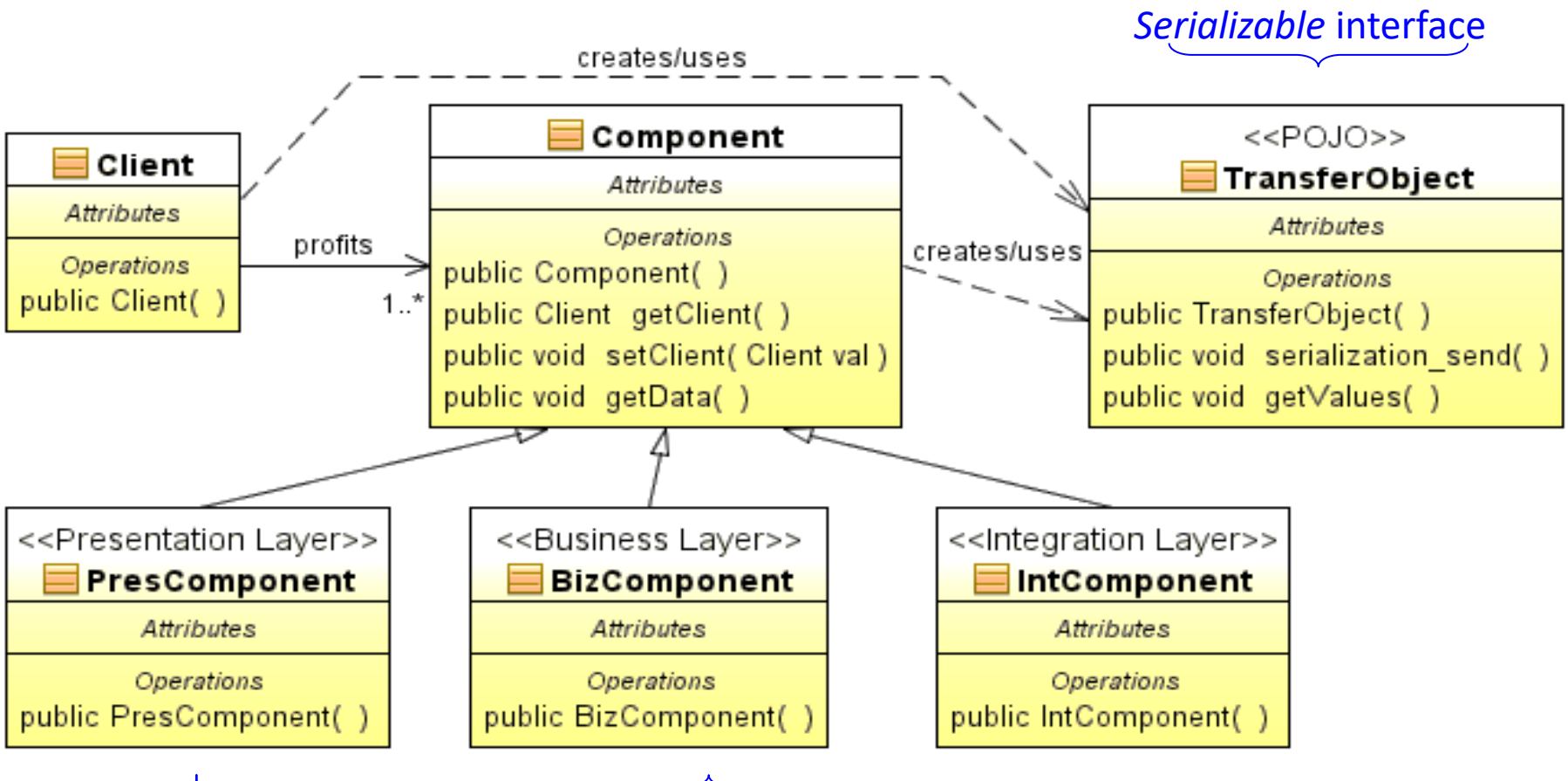


Komponenty EJB, *JMS* , lub część wzorca *SessionFacade*

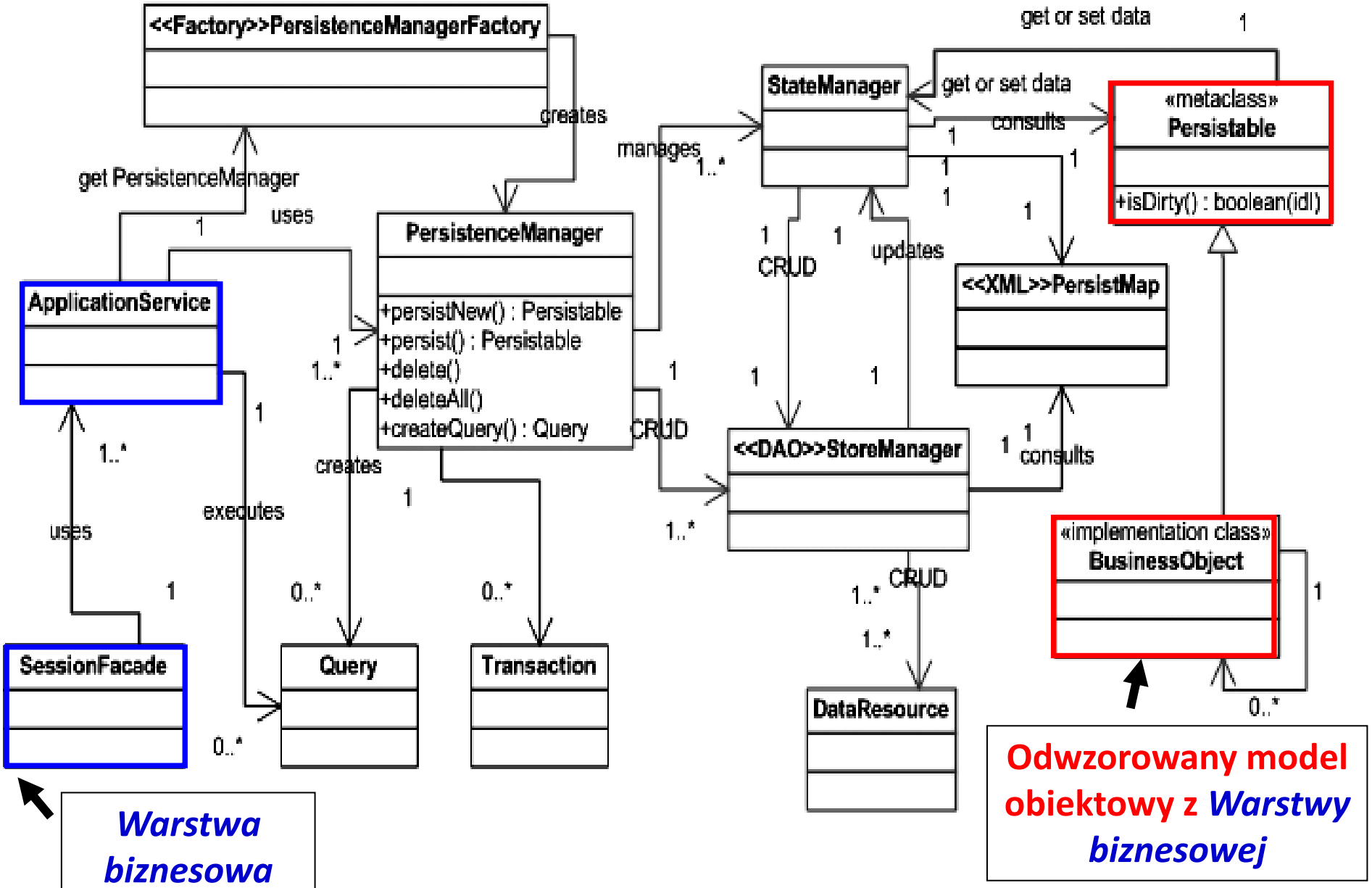
3.7. Wzorce EE *Warstwy biznesowej*: *SessionFacade, ApplicationService* – udostępnianie i centralizacja logiki biznesowej kilku komponentów i usług biznesowych



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.**

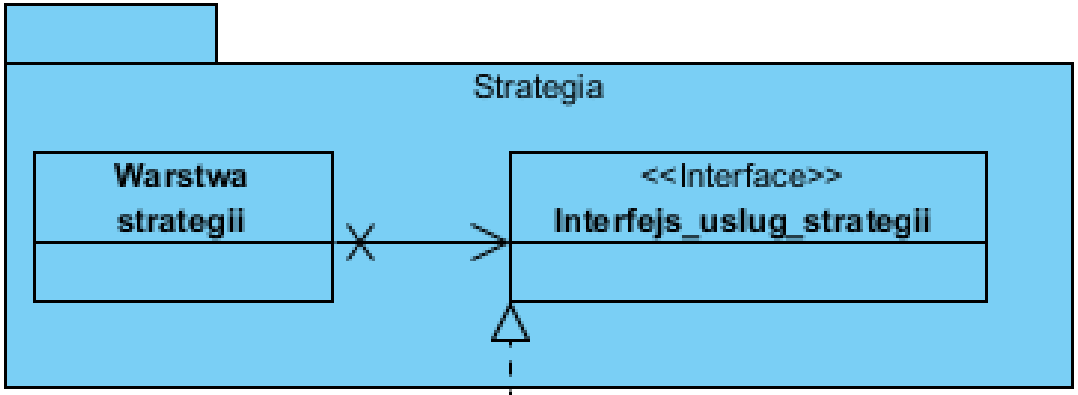
4. Zasada odwracania zależności (Dependency Inversion Principle –DIP)

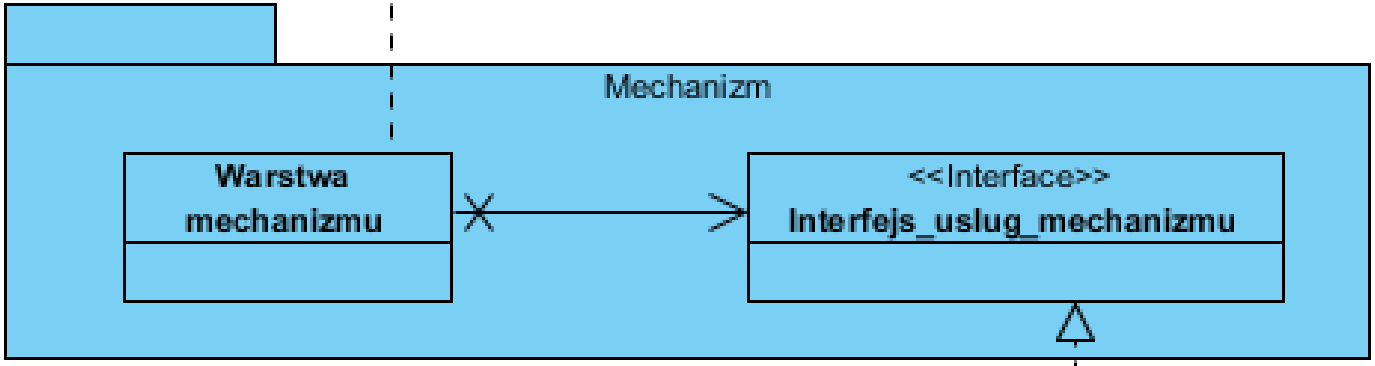
A. Moduły wysokopoziomowe nie powinny zależeć od modułów niskopoziomowych. Obie grupy modułów powinny zależeć od abstrakcji

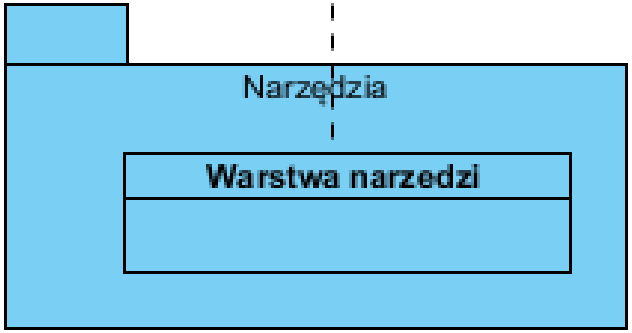
B. Abstrakcje nie powinny zależeć od szczegółowych rozwiązań. To szczegółowe rozwiązania powinny zależeć od abstrakcji.

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



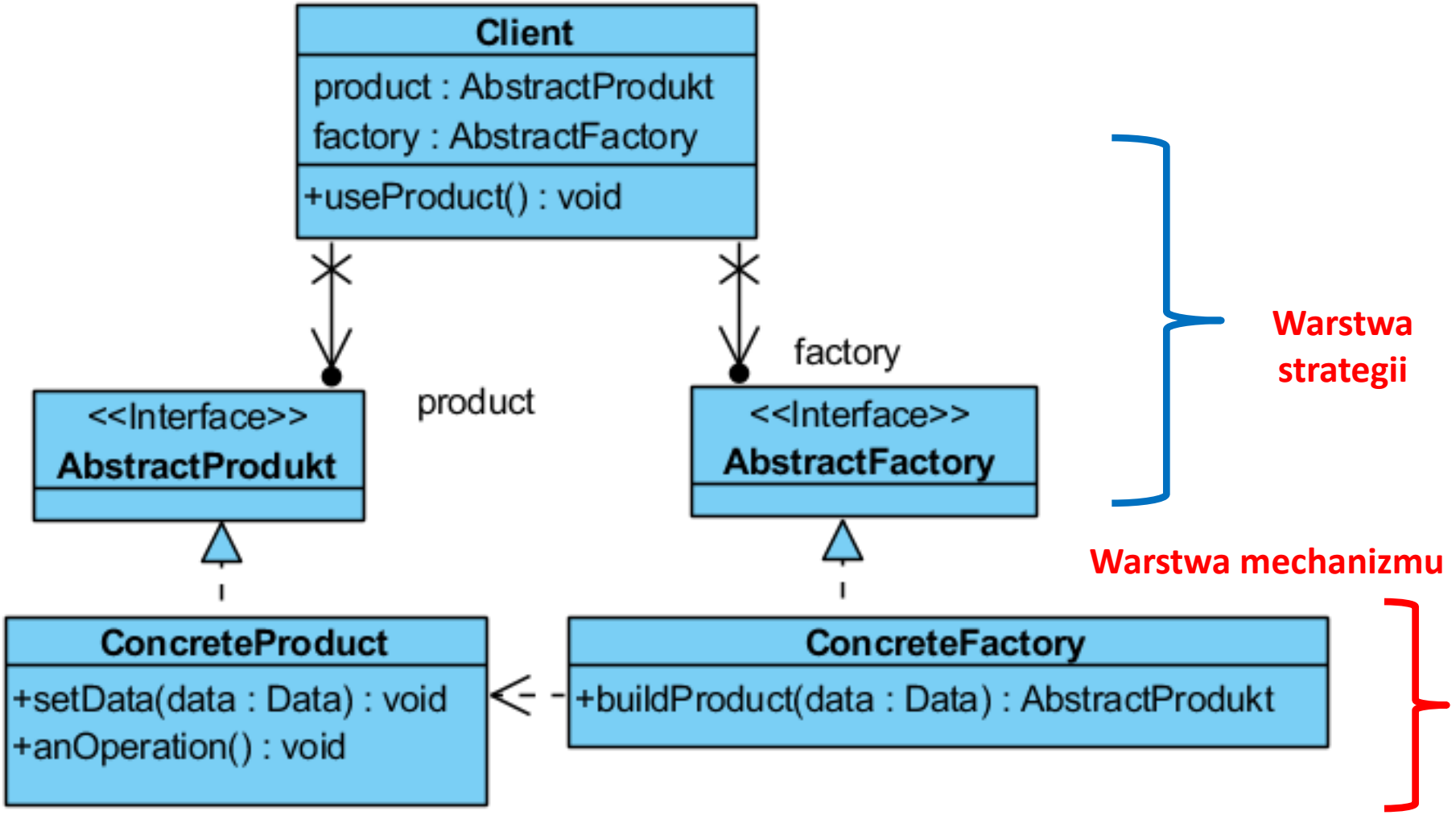




- Warstwy niższe zależą od wyższych
- Eliminacja zależności przechodniej:
  Warstwa strategii – warstwa narzędzi
- Eliminacja zależności bezpośredniej:
  Warstwa strategii – **Warstwa mechanizmu**

4 B- Zasady procesu solidnego - przykład

Fabryka abstrakcyjna – *Abstract Factory*



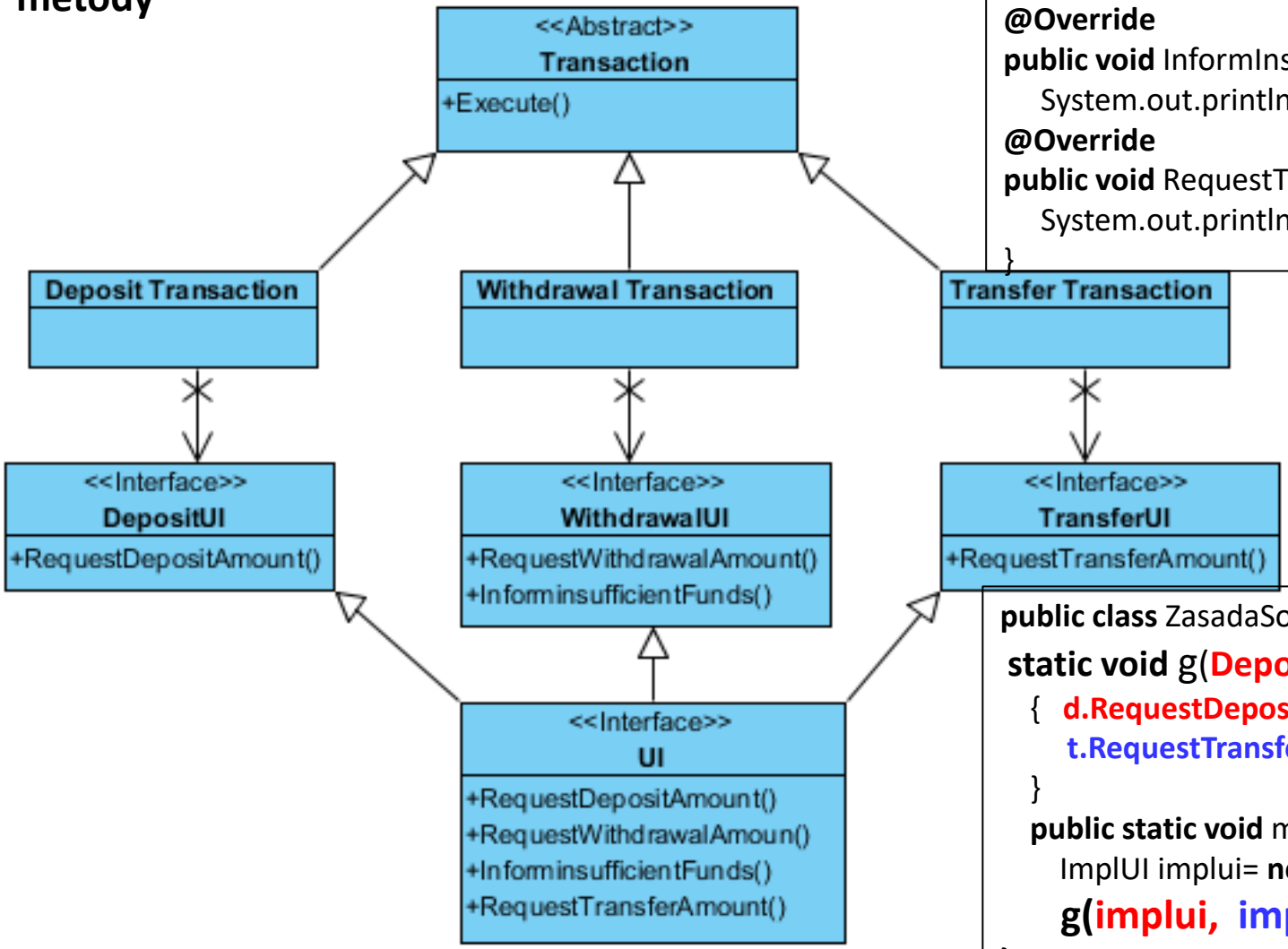
5. Zasada segregacji interfejsów (Interface Segregation Principle – ISP)

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]

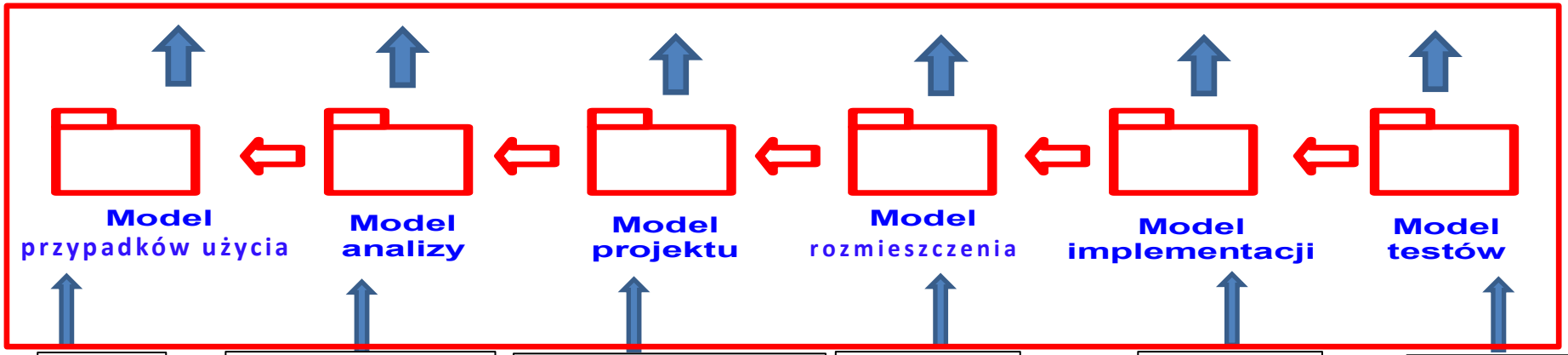
- Kluczowy mechanizm niskiego poziomu
- Podwyższa odporność kodu na zmiany
- Prowadzi do tworzenia kodu wielokrotnego użycia

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 *co należy wykonać?*	Perspektywa specyfikacji *jak należy używać?*	Perspektywa implementacji *jak należy wykonać?*
• model problemu np. przedsiębiorstwa • wymagania • analiza (model konceptualny: diagram przypadków użycia, diagram klas, diagramy sekwencji,) • testy modelu	• projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • testy projektu	• programowanie, wdrażanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • testy oprogramowania • wdrażanie • testy wdrażania



2.1, 2.2

1.1, 1.2, 1.3, 2.5, 2.7, 2.3 – wyższy poziom abstrakcji niż w modelu projektowym

1.1, 1.2, 1.3, 2.5, 2.7, 2.3-więcej szczegółów niż w modelu analizy (niższy poziom abstrakcji)

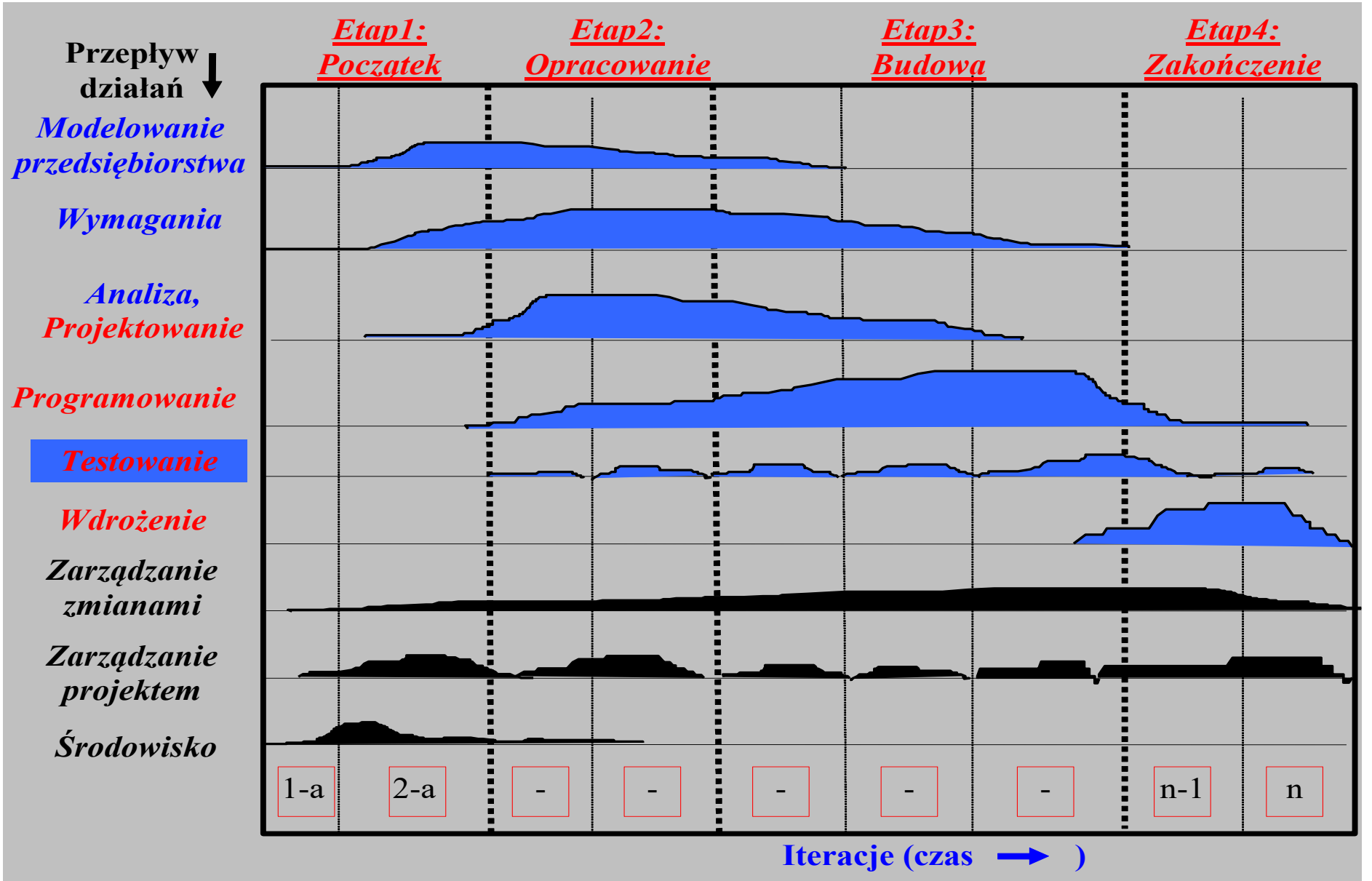
1.6, 1.5

1.2, 1.5

1.2, 2.2

Numery diagramów UML: (**wykład 1**)

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 zawierajacą 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 wypozyczenia 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 wypozyczenia 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 przeglądającdających 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 niefunkcjonalnych

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 kaseta, 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 kaseta oraz numer egzemplarza.
6. Zarówno egzemplarze typu książka lub kaseta, mogą być przeznaczane do wypożyczenia na okres umowny oraz na okres ściśle określony.

Lista wymagań niefunkcjonalnych 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

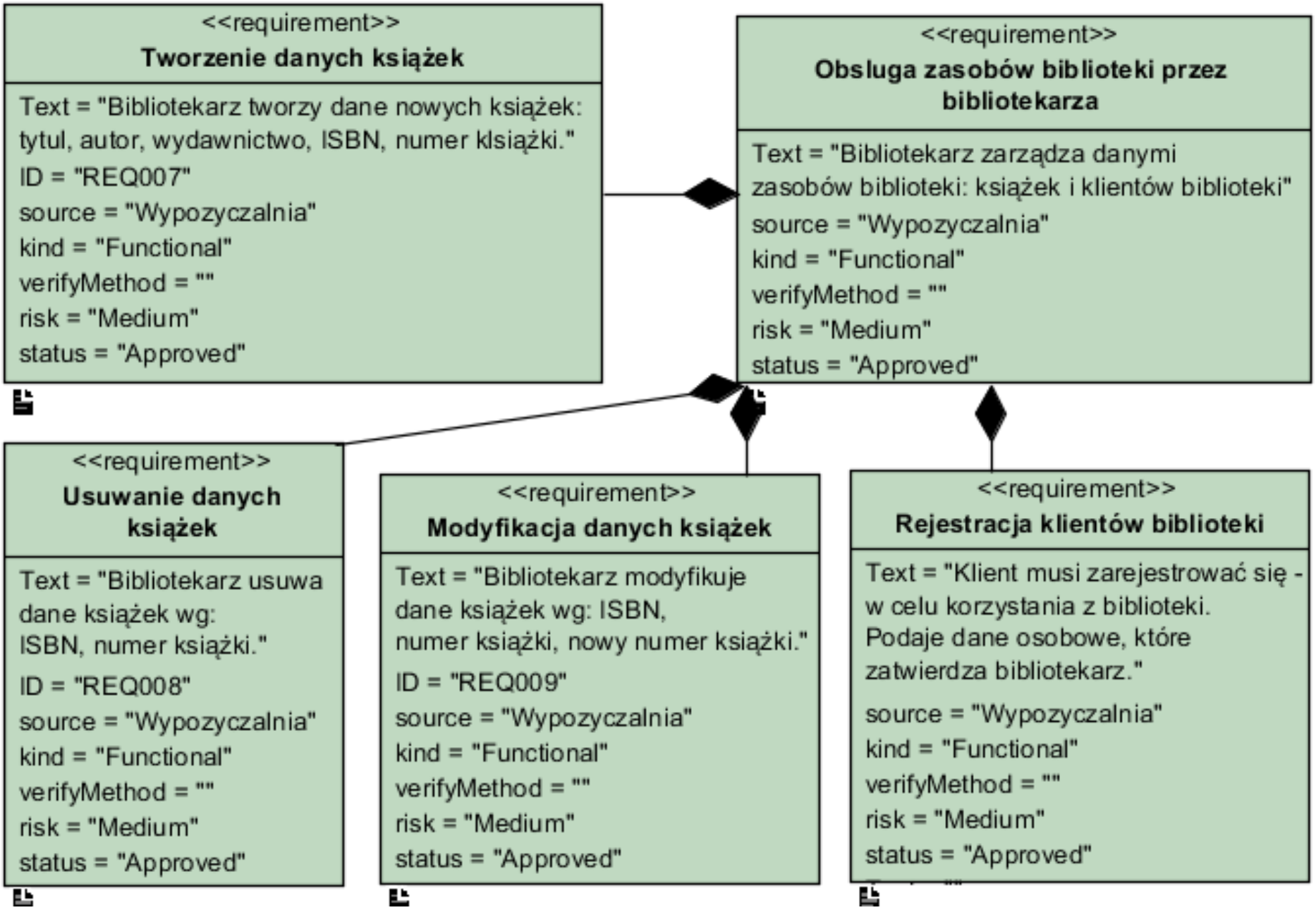


Diagram wymagań funkcjonalnych programu(cd)

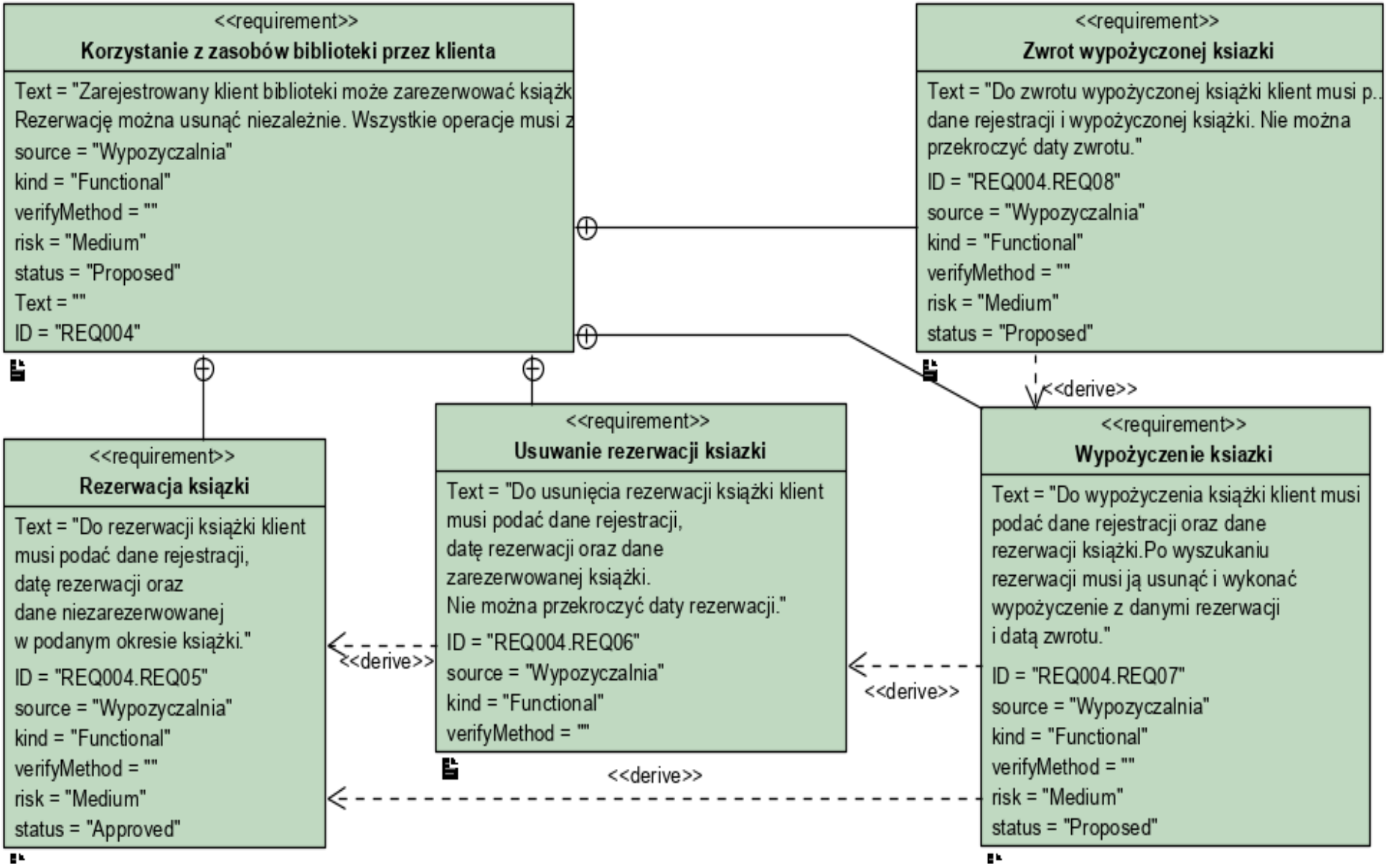
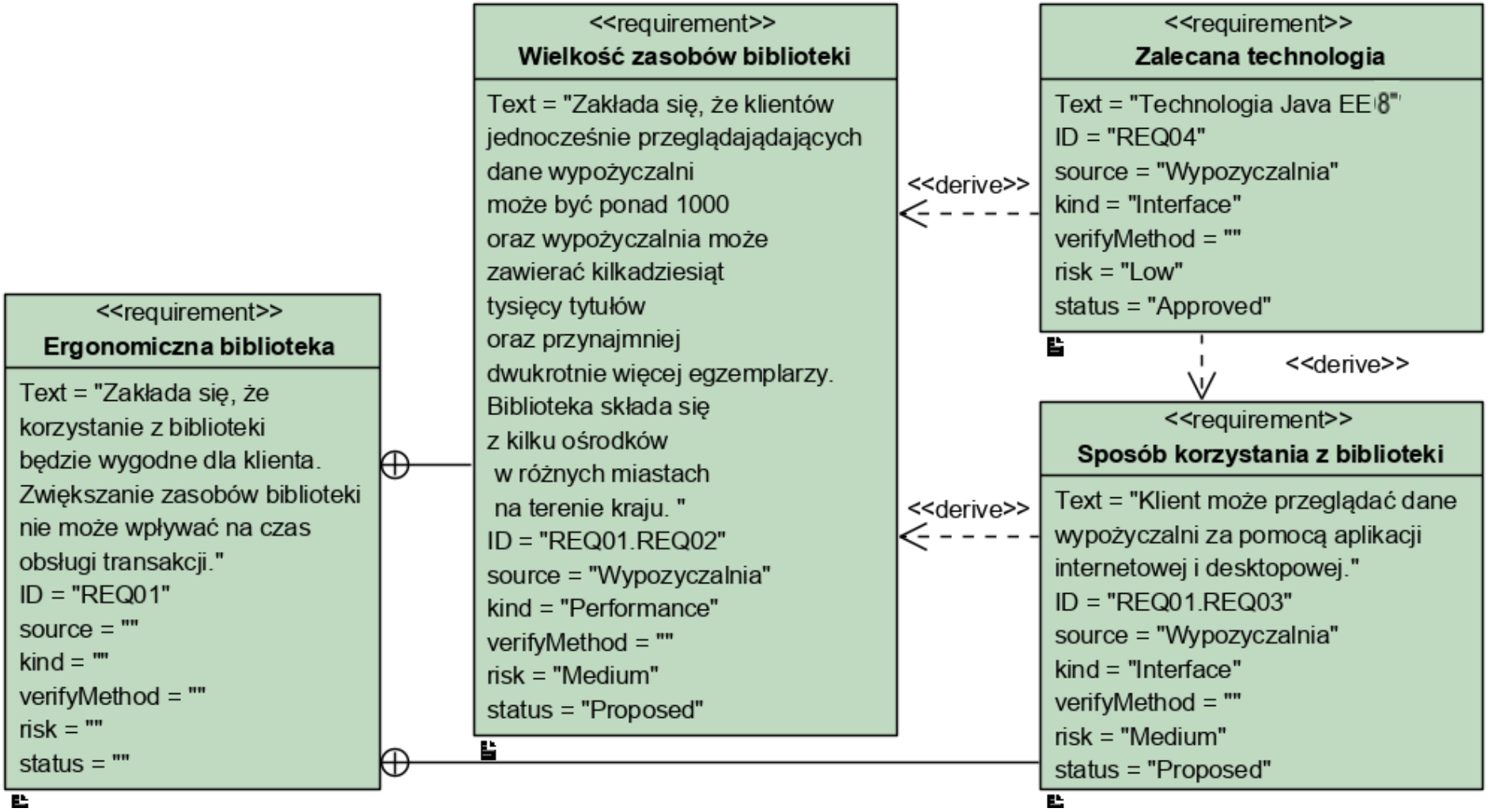


Diagram wymagań niefunkcjonalnych programu (cd)



III. Model analizy aplikacji oparty na diagramie przypadków użycia

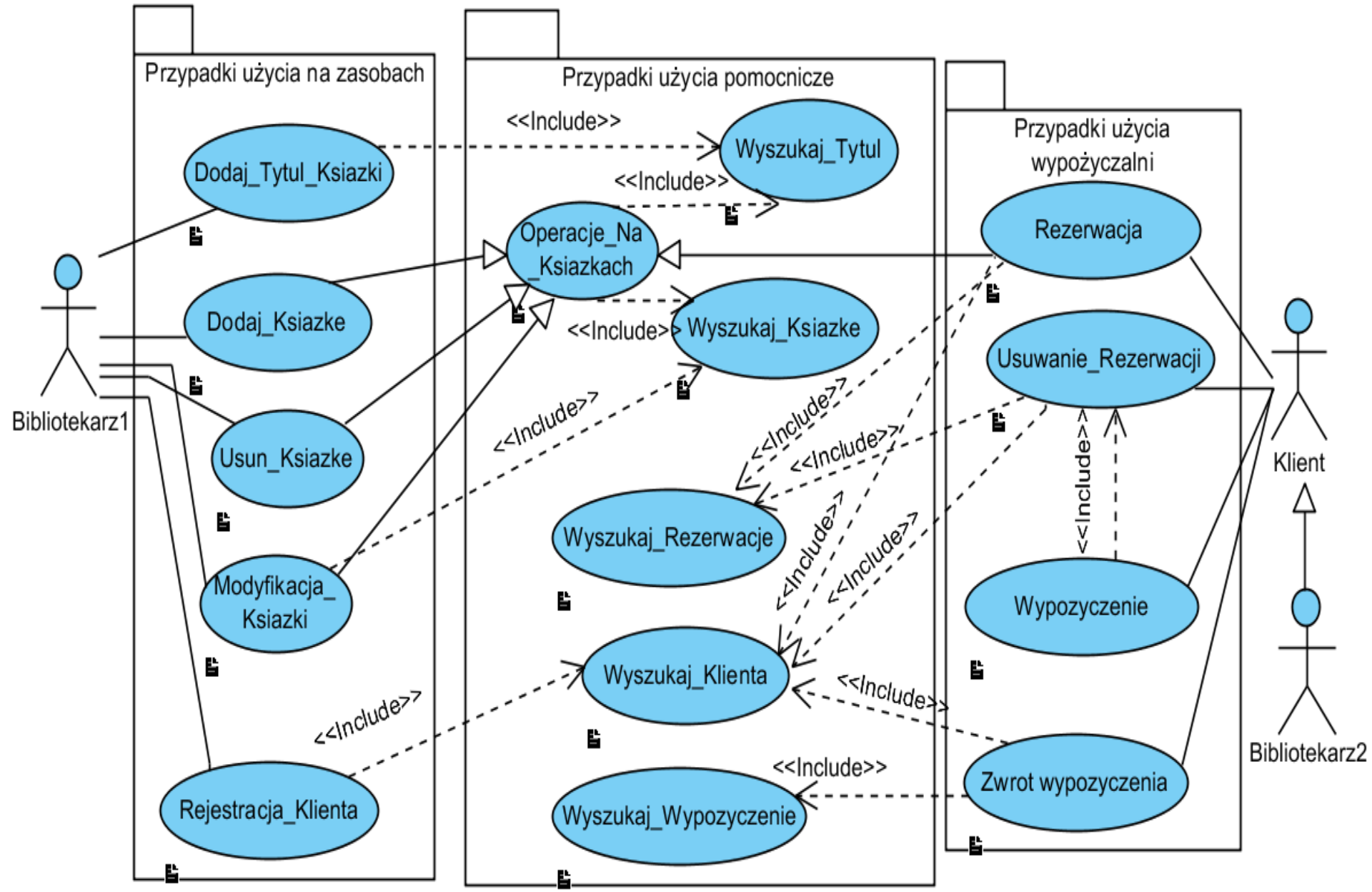
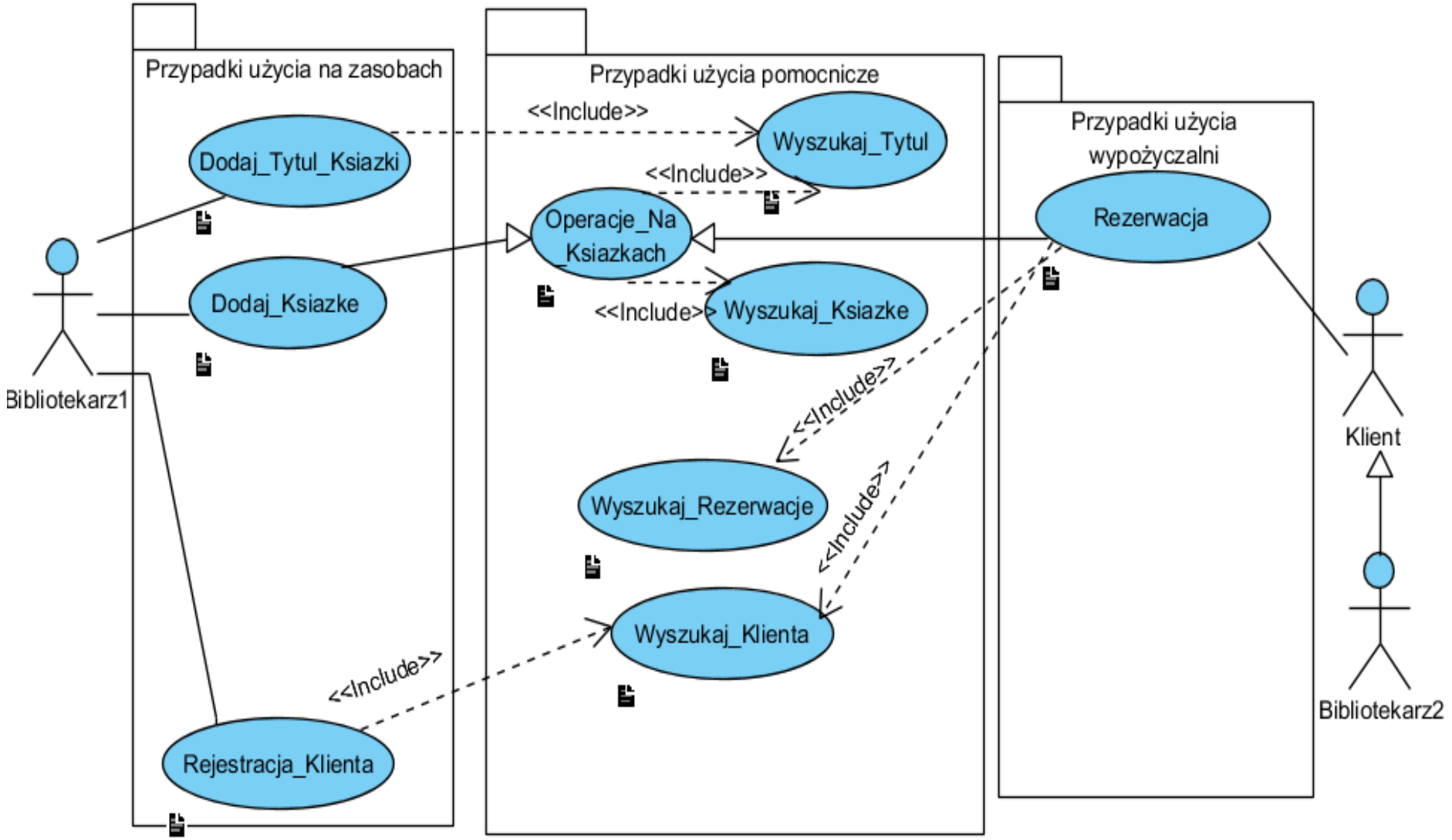


Diagram przypadków użycia - wybrany fragment bez realizacji wypożyczenia



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

Podejście zaprezentowane w instrukcji do laboratorium 1 -

http://zofia.kruczkiewicz.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)

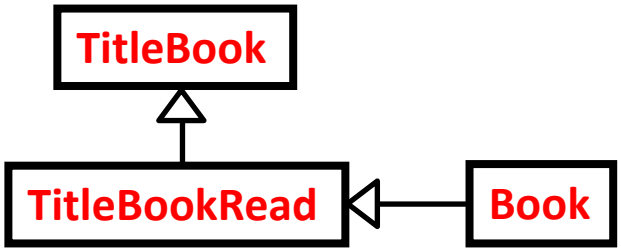
- Wykryto **dziedziczenie** w właściwościach tytułów książek, które określają, które książki są nagrane, a które są drukowane Zdefiniowano klasę pochodną:
  - **TitleBookRead** typu **Entity**, która dziedziczy od klasy **TitleBook** **i zawiera informację o aktorze, która czyta książkę na nagraniu** (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 tyupu **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 dwukierunkowejmiędzy 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 nagraną, czy też papierową (**PU Dodaj_Tytul**, **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**

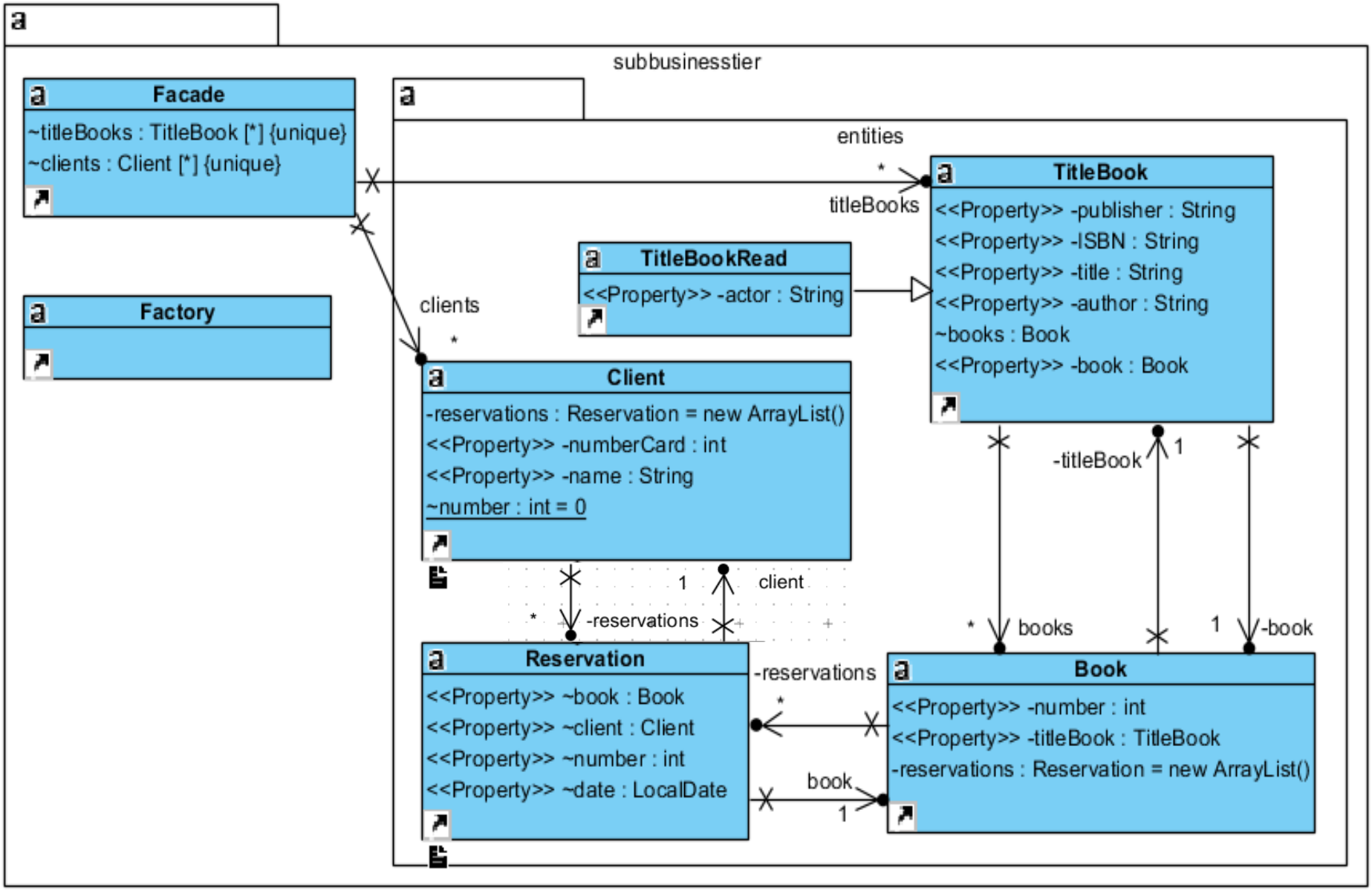
Analiza zmienności (c.d)

- **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**

Wynik

Początkowa definicja diagramu klas – zdefiniowano powiązania między klasami



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.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/IO_UML/Instrukcja_1_2.pdf

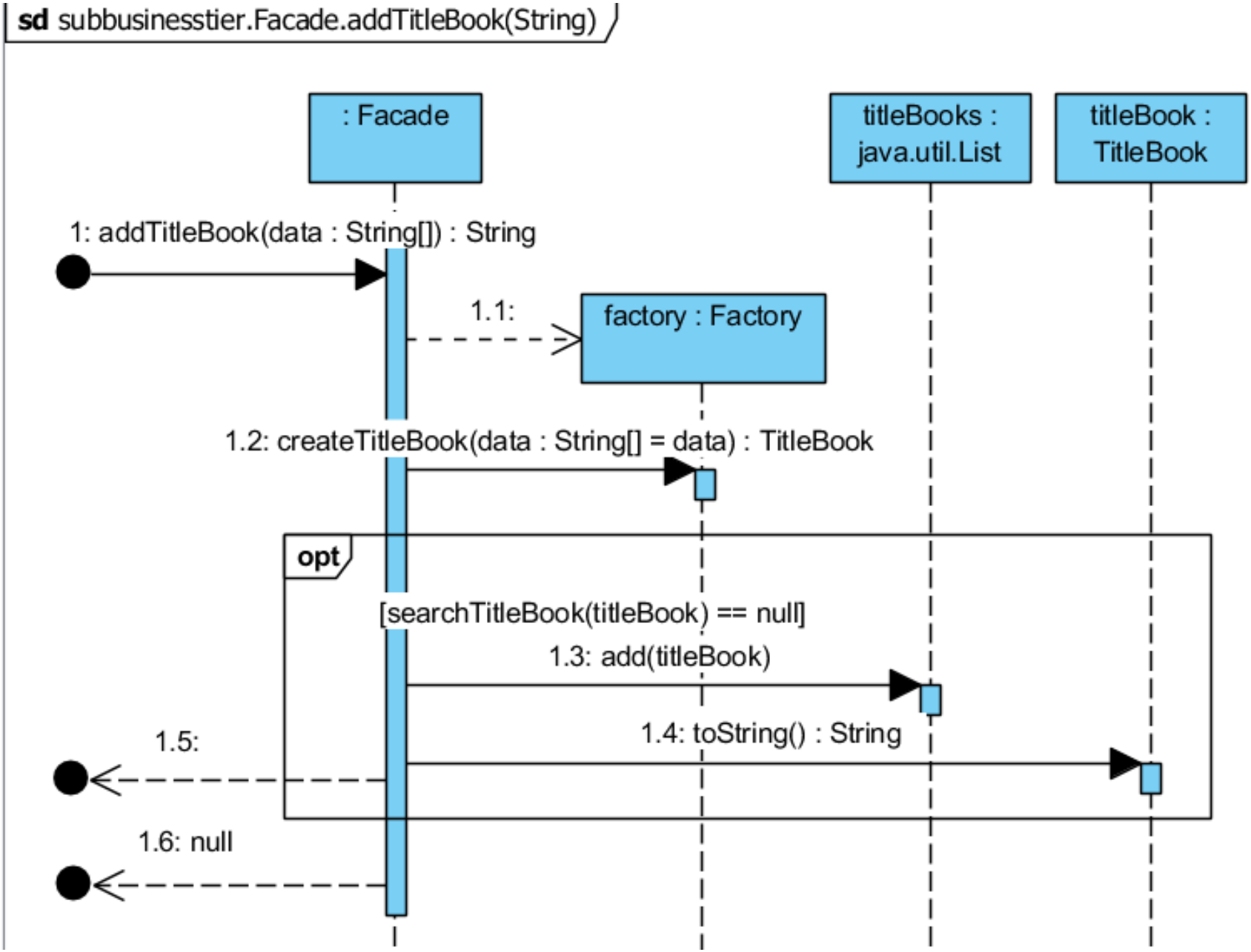
Iteracja 1

Projekt przypadku użycia

„ **Dodaj_Tytul_Ksiazki**”

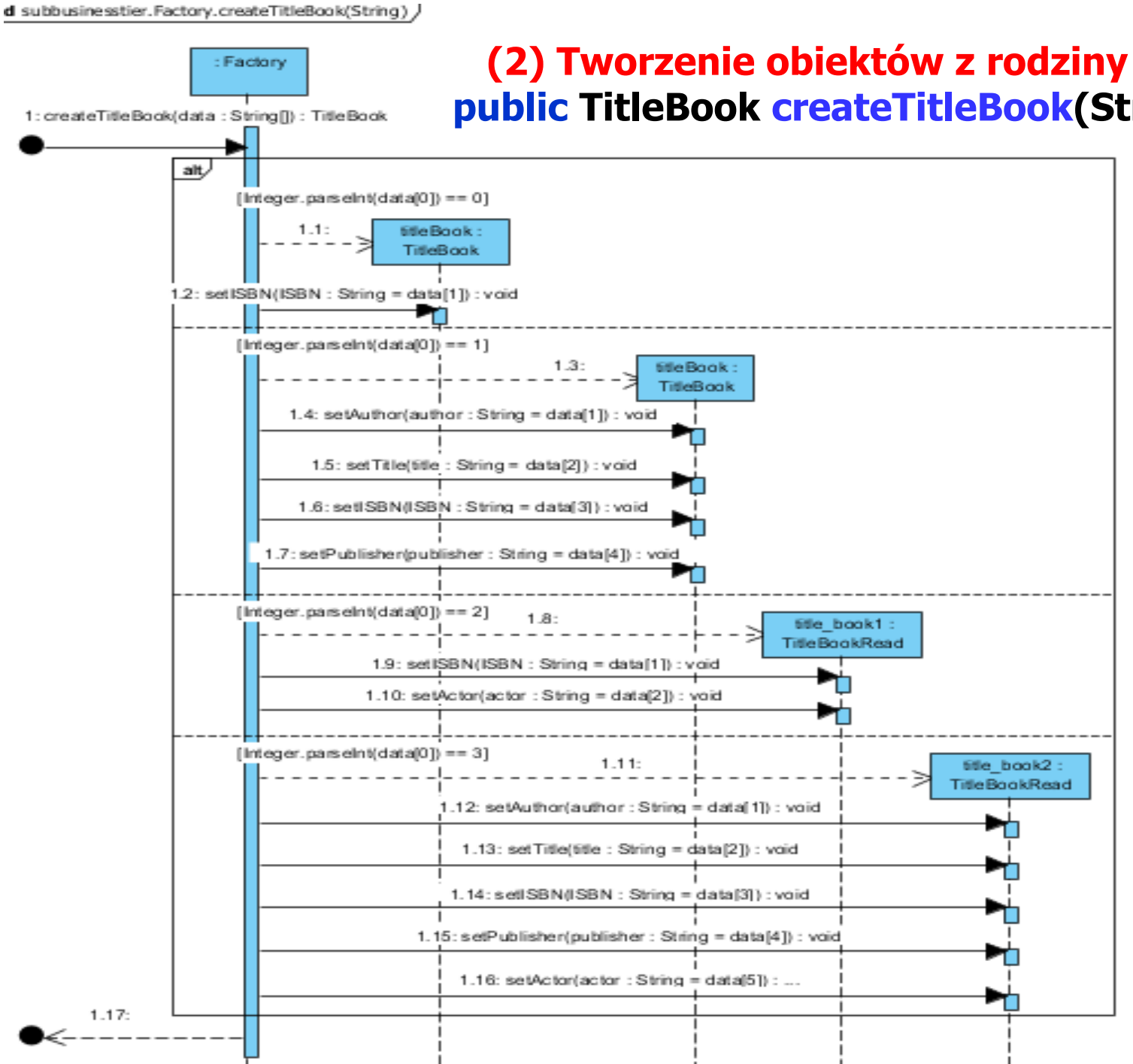
za pomocą diagramu sekwencji i diagramu klas. Diagram klas jest uzupełniany metodami zidentyfikowanymi podczas projektowania scenariusza przypadku użycia za pomocą diagramu sekwencji.

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



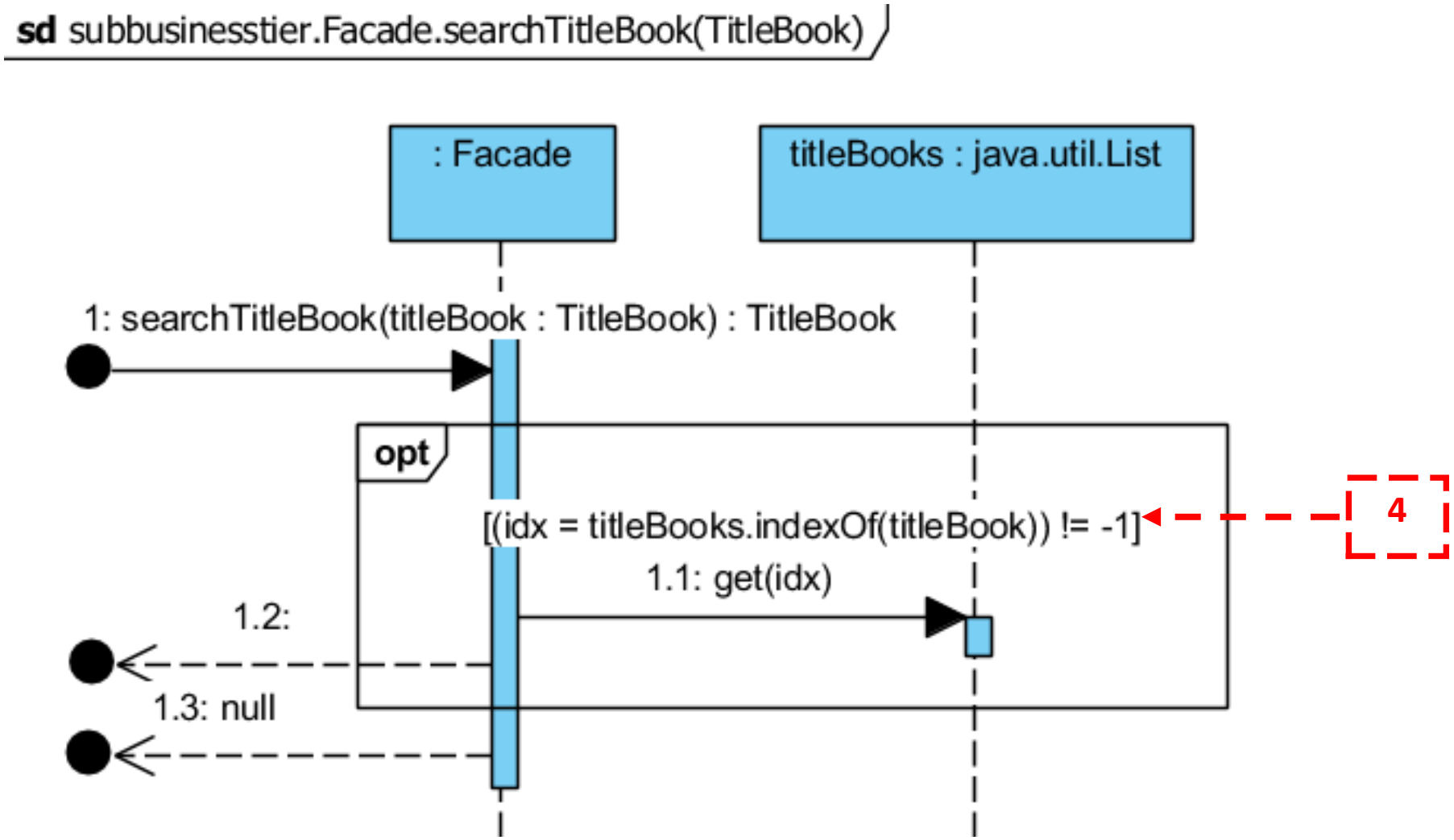
(2) Tworzenie obiektów z rodziny TitleBook

public TitleBook createTitleBook(String data[])

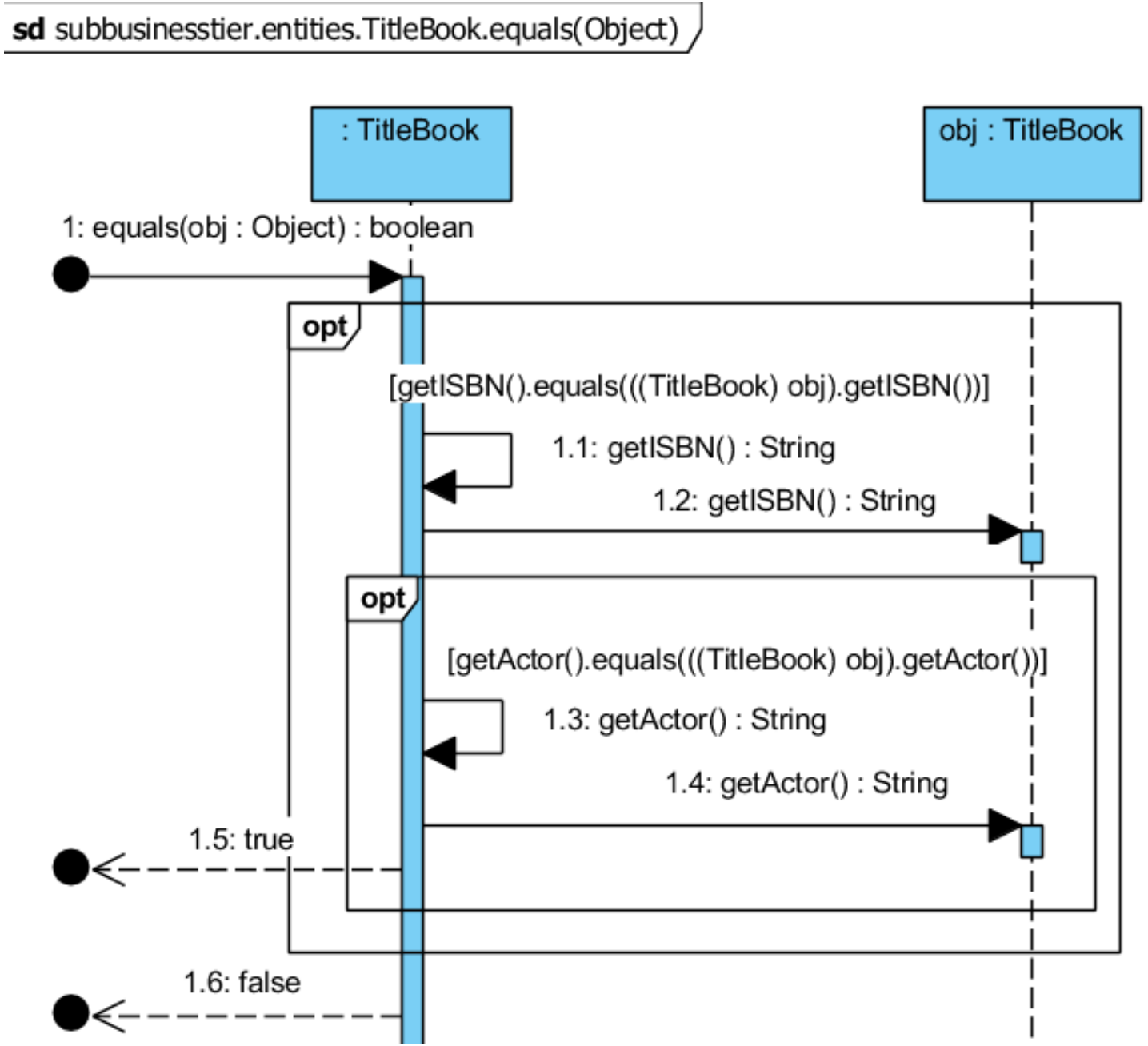


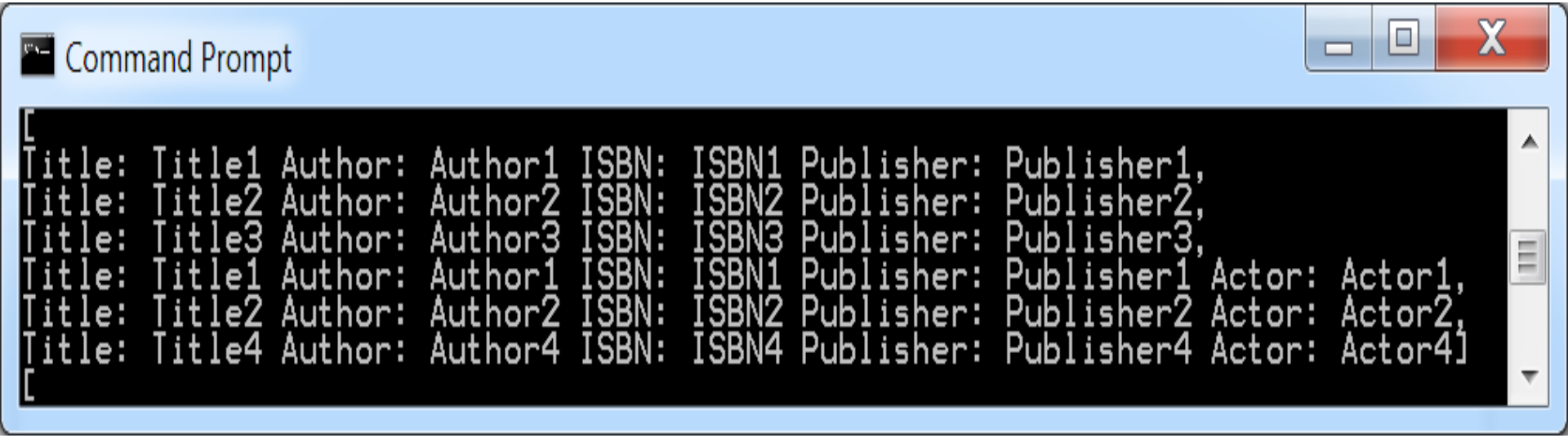
(3) Wyszukiwanie obiektu z rodziny TitleBook

public TitleBook searchTitleBook(TitleBook titleBook)



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




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]
[

Iteracja 2

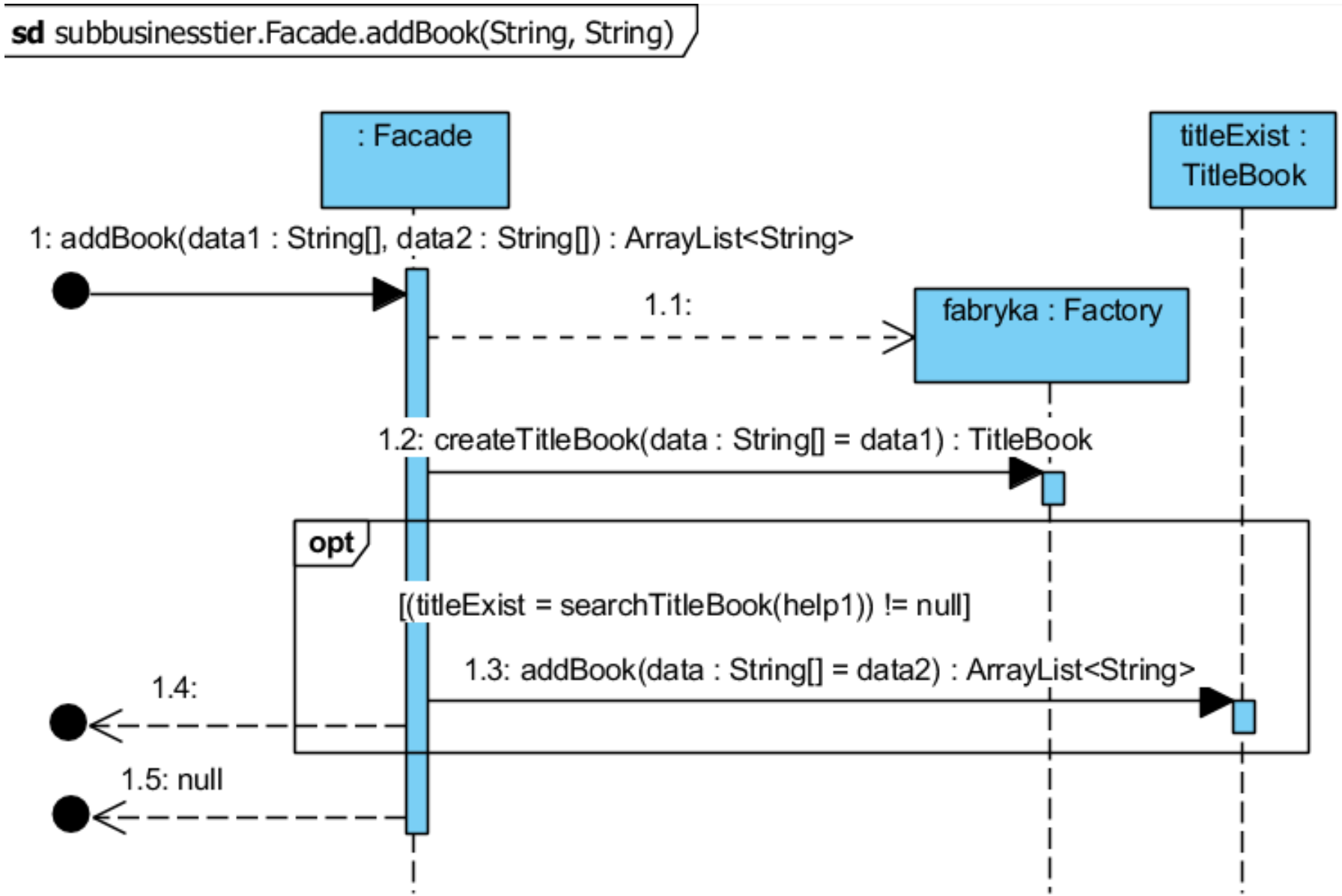
Projekt przypadku użycia

„**Dodaj_Ksiazke**”

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

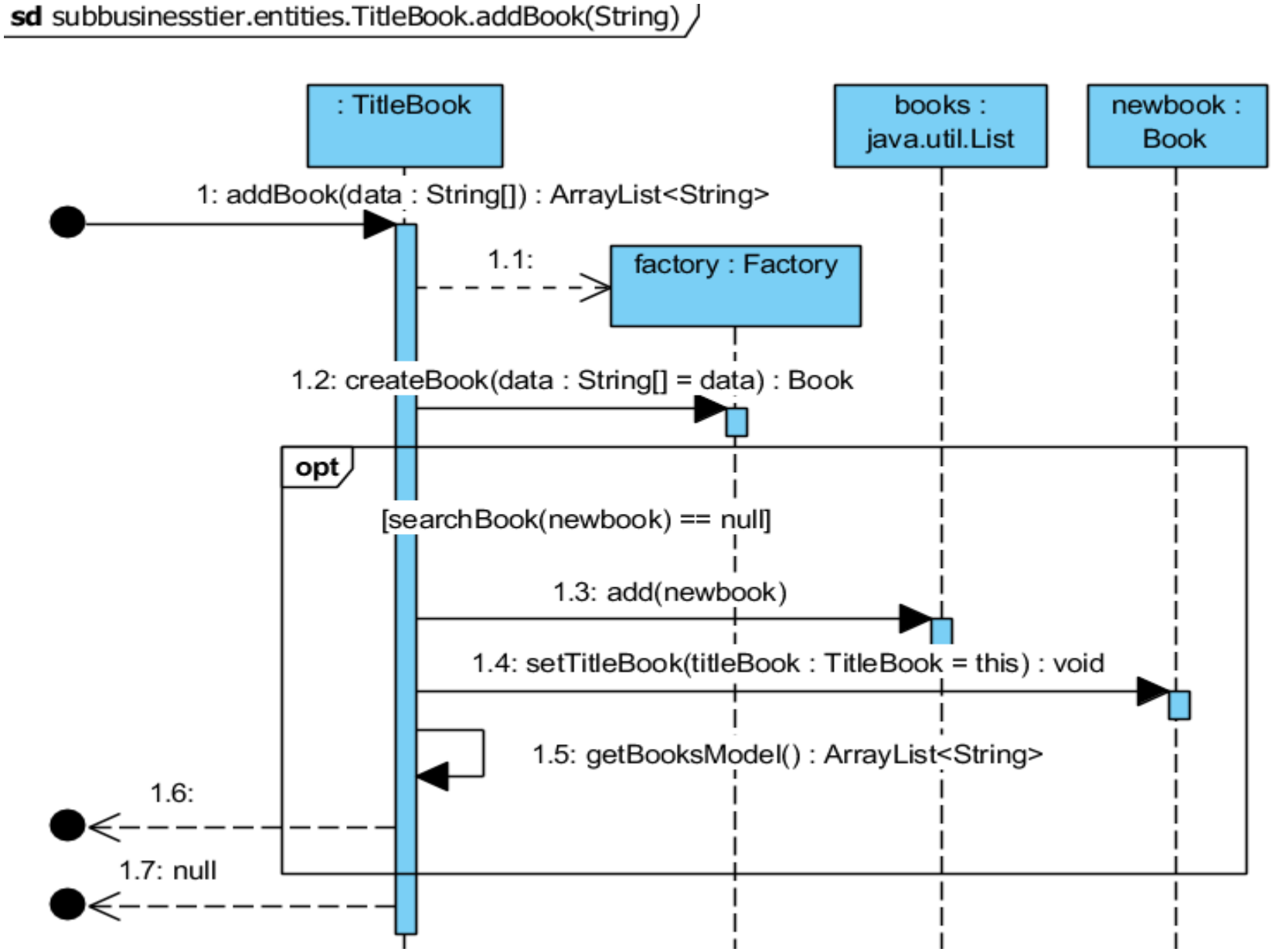
(5) Wstawianie nowej książki o podanym tytule – 1-y etap

public ArrayList<String> addBook(String data1[], String data2[])



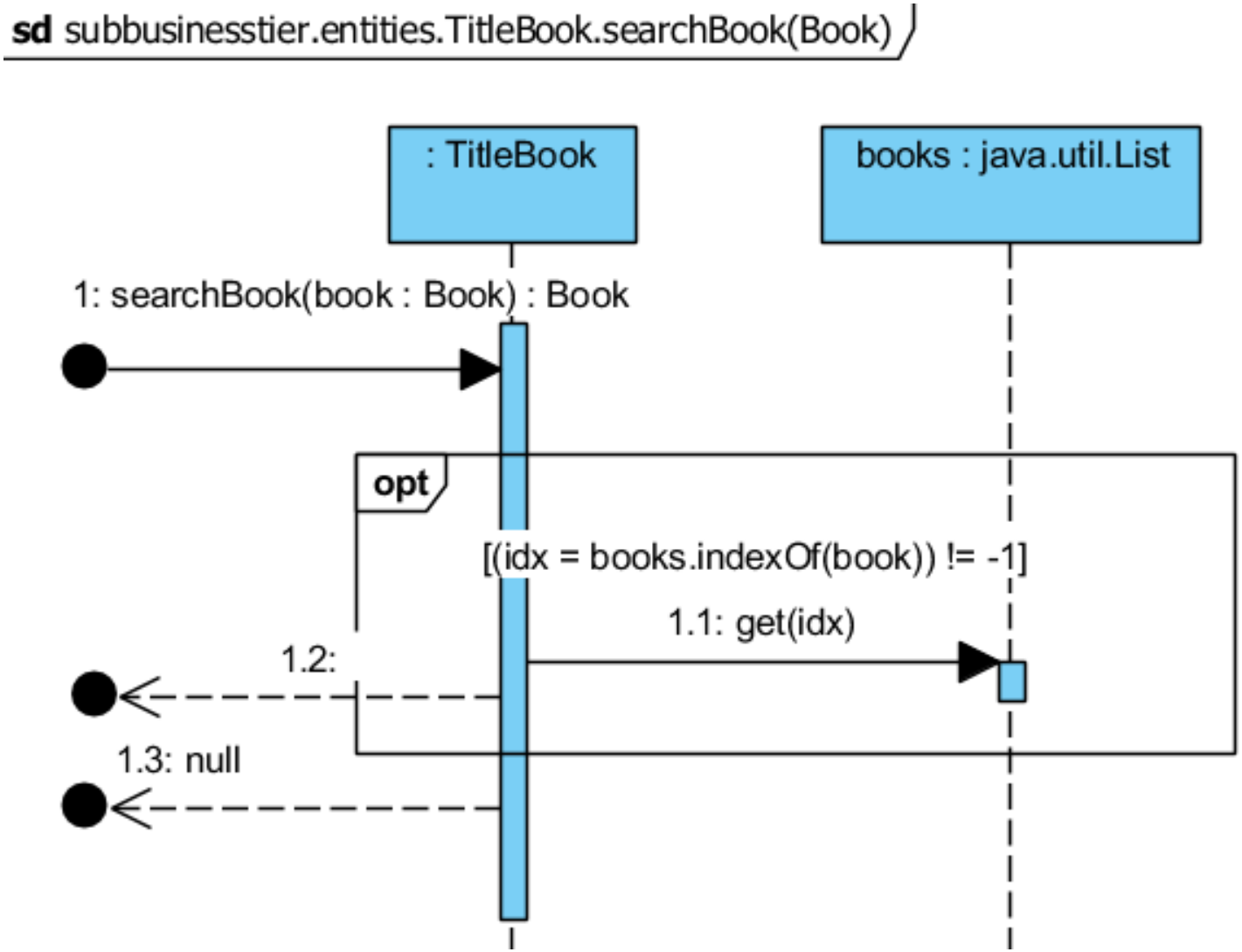
(6) Wstawianie nowej książki o podanym tytule – 2-i etap

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



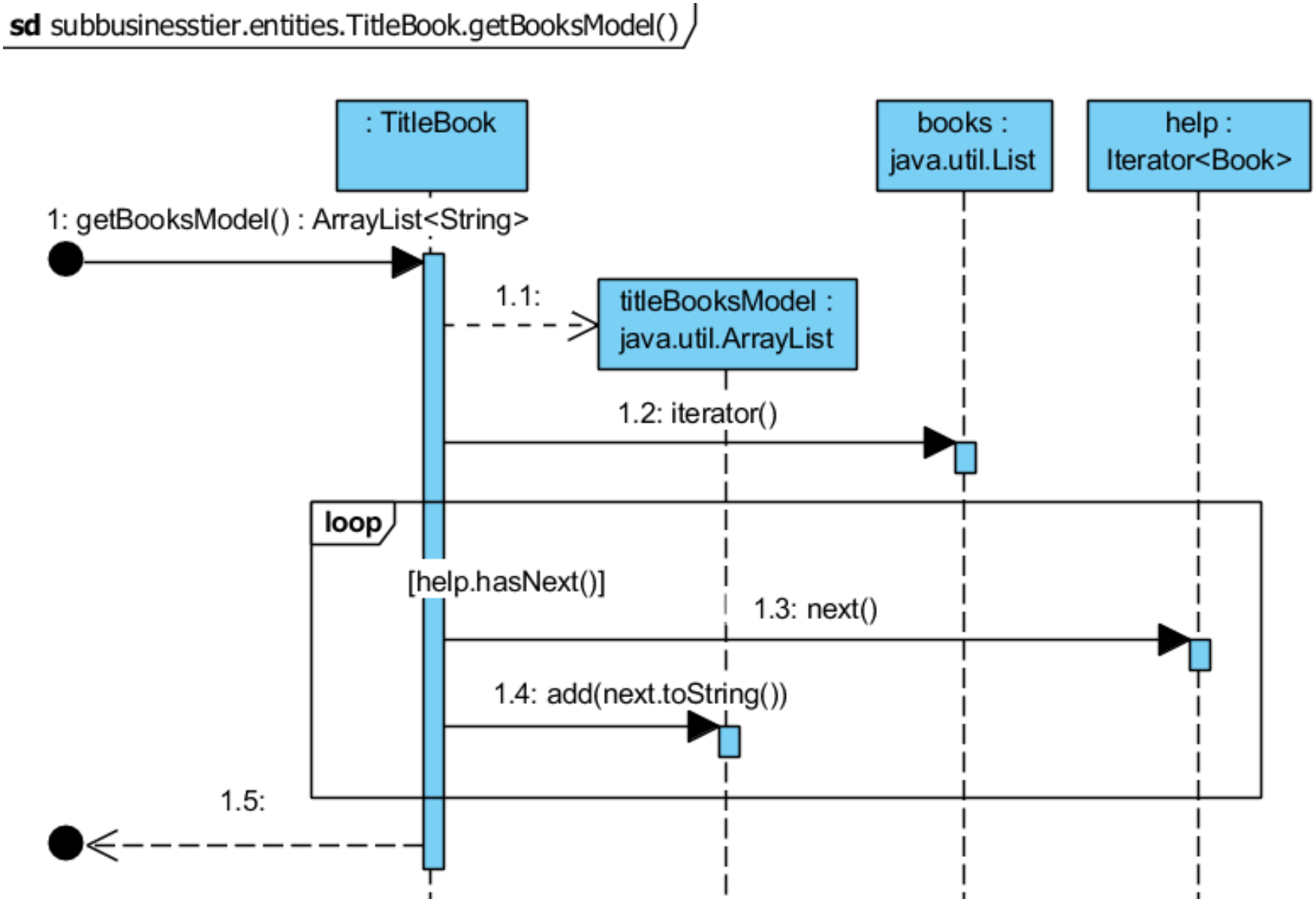
(7) Szukanie książki

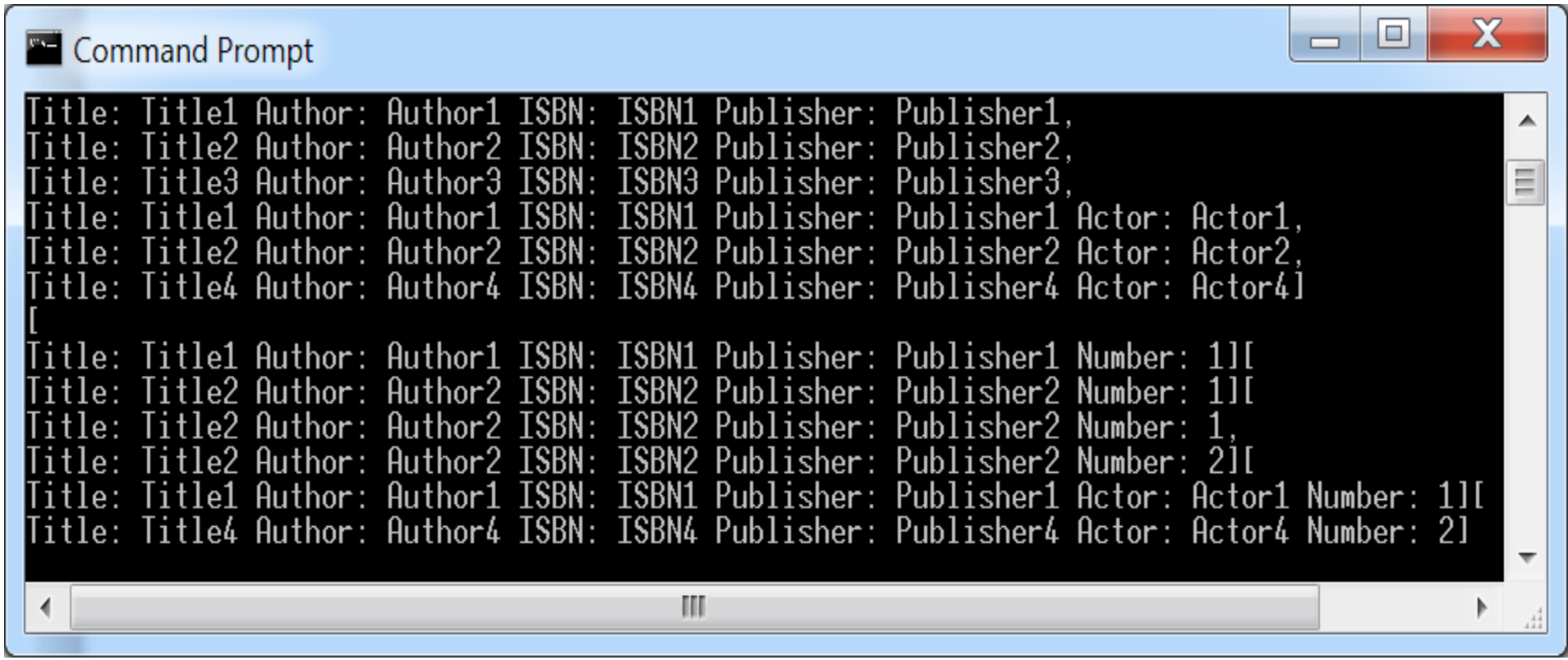
public Book searchBook(Book book)



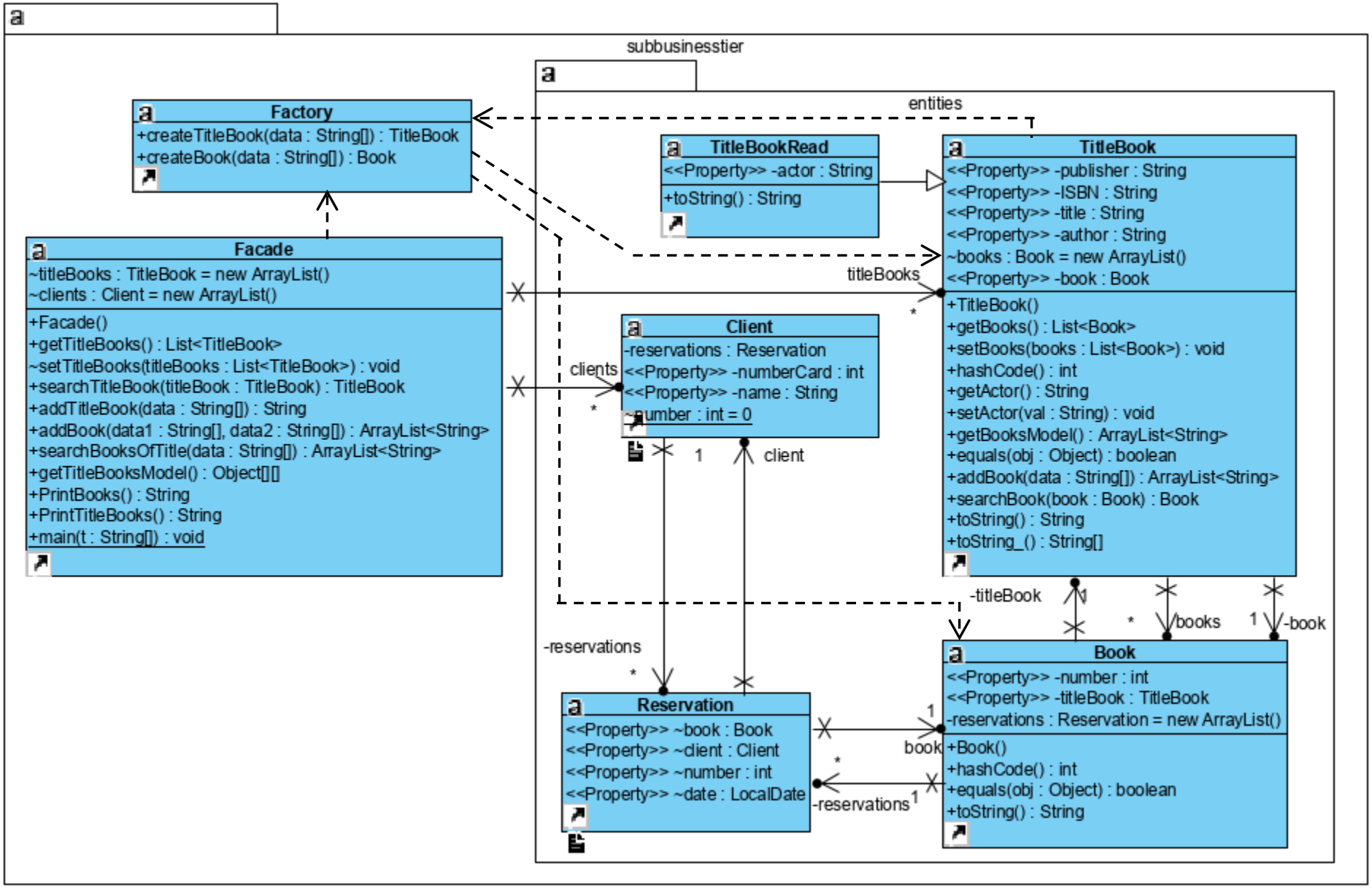
(8)Pobranie danych o książkach należących do obiektu z rodziny TitleBook

public ArrayList<String> **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

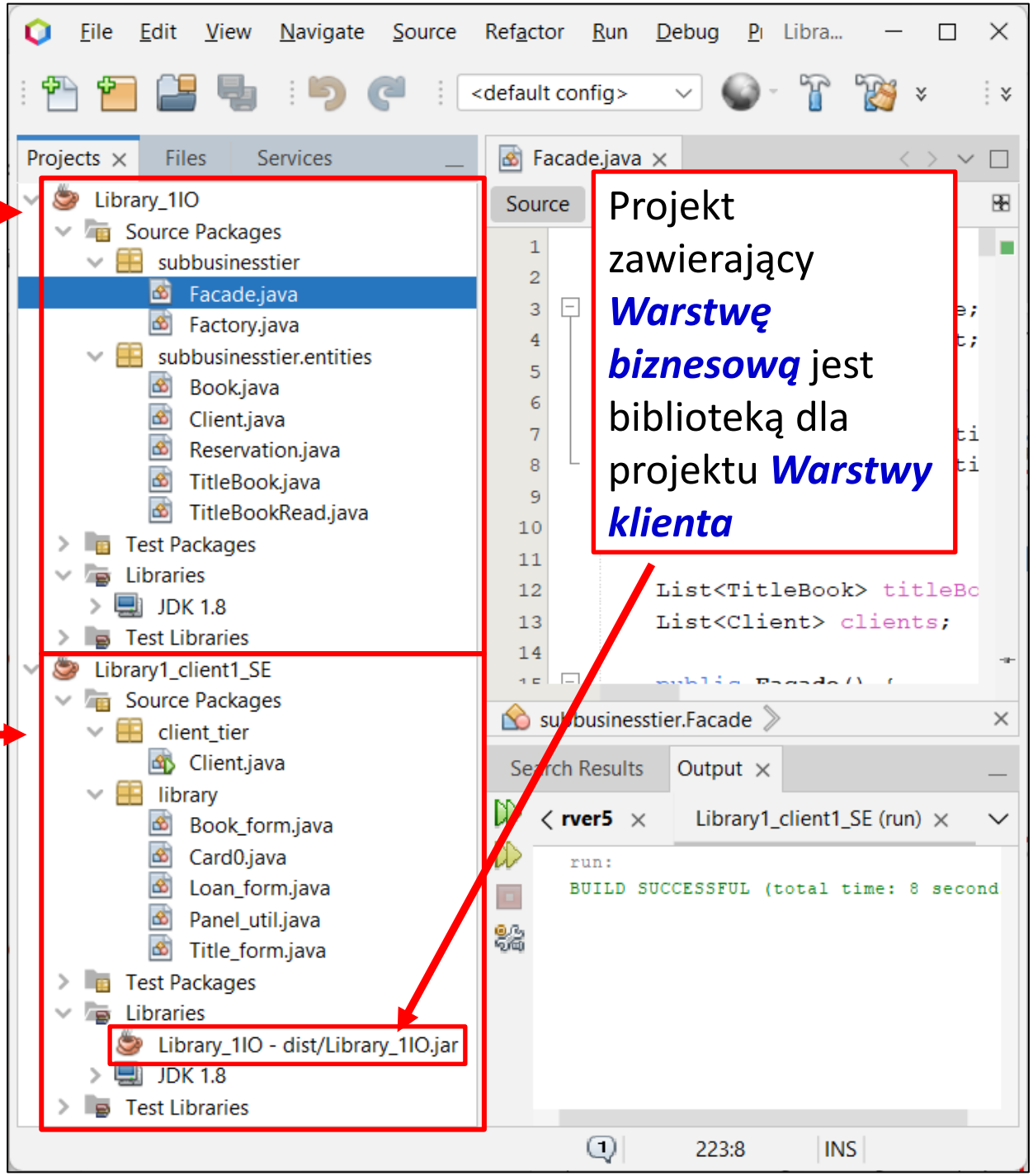
Warstwa klienta
-aplikacja desktopowa (GUI) do wykonania
Library1_client1_SE

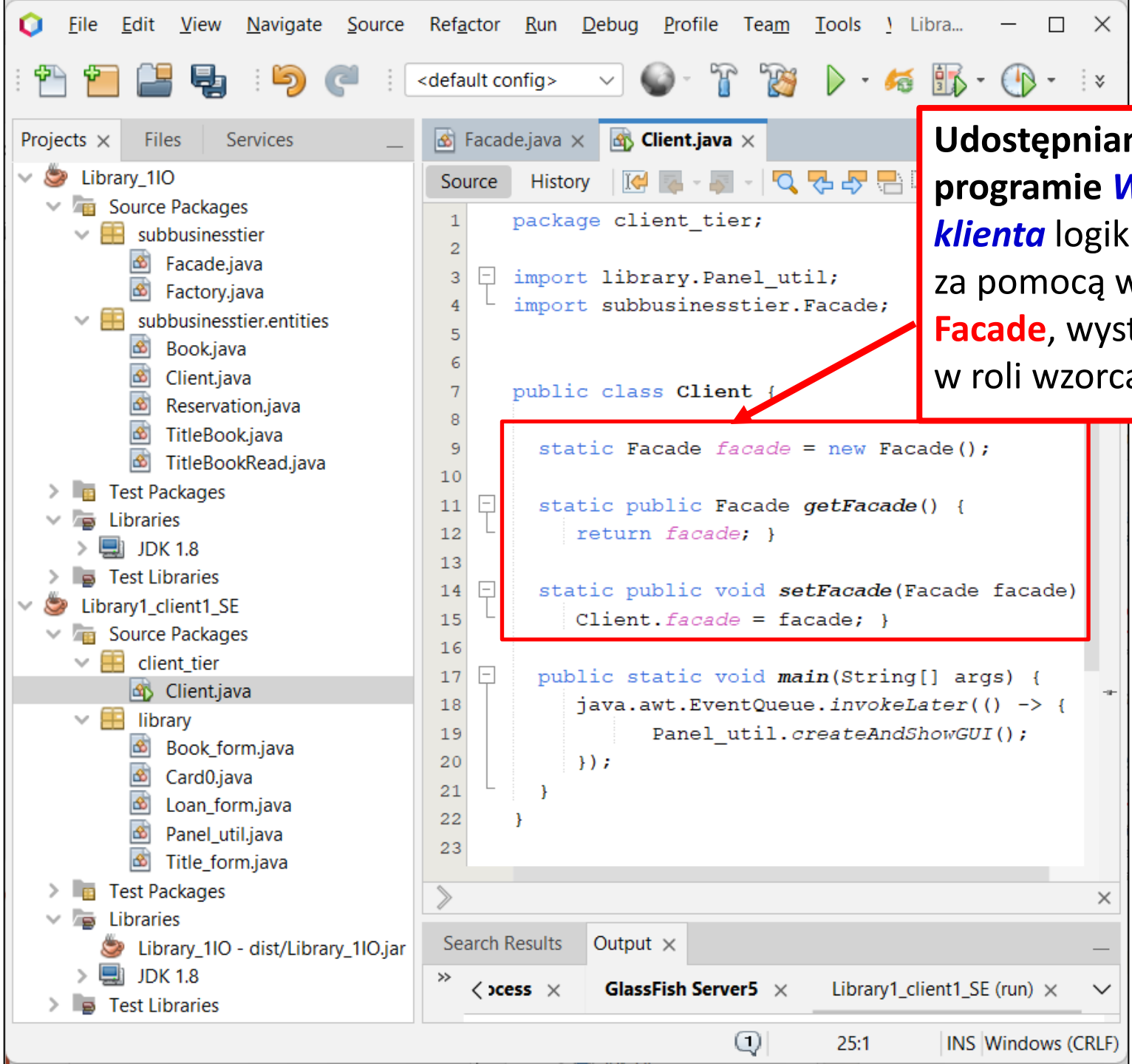
***Warstwa biznesowa* – przetwarzanie danych. Projekt typu Java Class Library, używany jako biblioteka logiki biznesowej, który zawiera kod utworzony podczas 2 iteracji- Library_1IO**

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

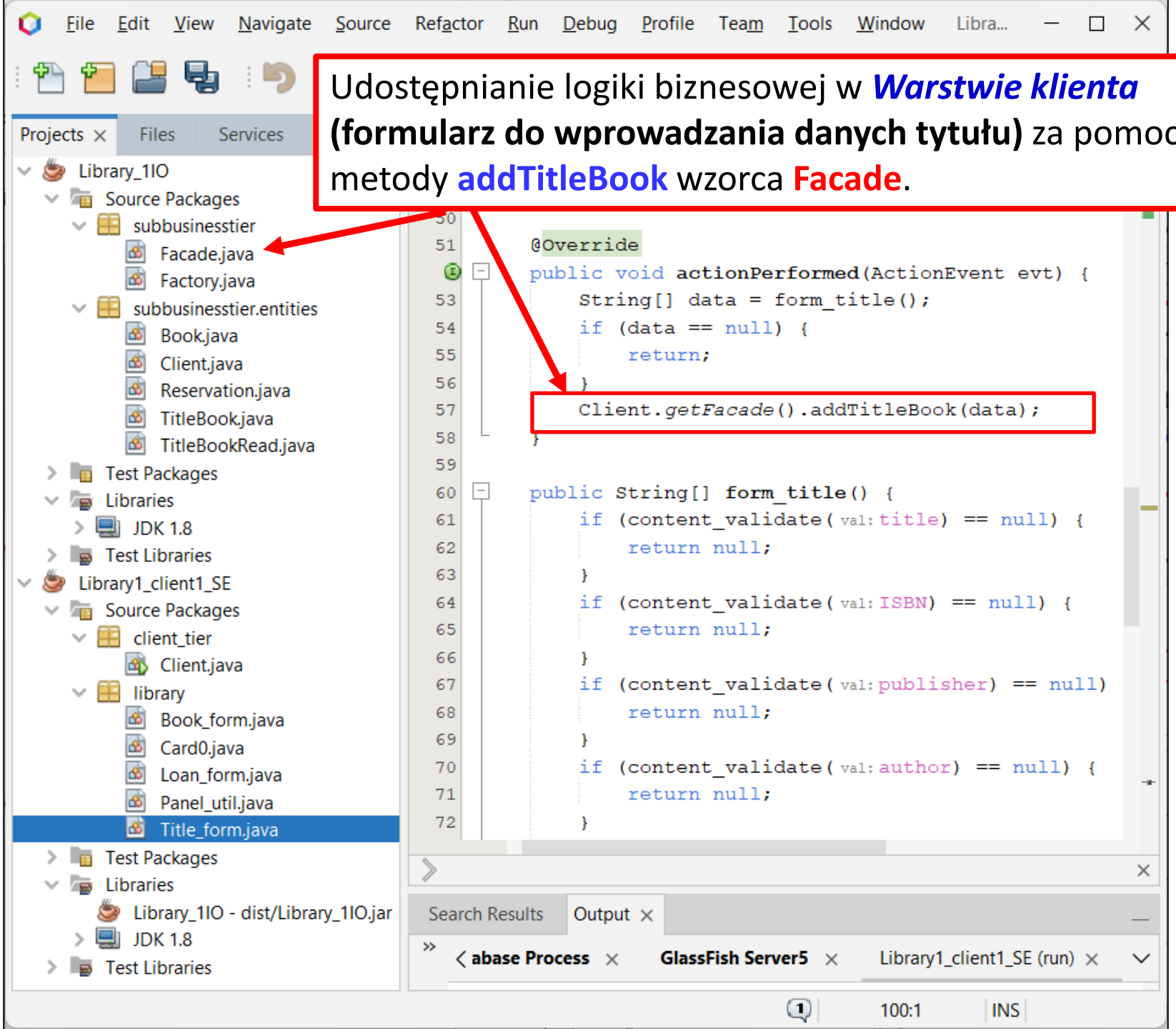
Projekt typu Java Class Library zawierający kod *Warstwy biznesowej* wykonany podczas dwóch iteracji

Projekt typu Java Application zawierający kod *Warstwy klienta* desktopowego z interfejsem graficznym użytkowania (GUI) do kodu 1-ej i 2-ej iteracji





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


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window
Udostępnianie logiki biznesowej w ***Warstwie klienta*** **(formularz do wprowadzania danych tytułu)** za pomocą metody **addTitleBook** wzorca **Facade**.
Projects
Files
Services
Library_1IO
Source Packages
subbusinesstier
Facade.java
Factory.java
subbusinesstier.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_1IO - dist/Library_1IO.jar
JDK 1.8
Test Libraries
@Override
public void actionPerformed(ActionEvent evt) {
String[] data = form_title();
if (data == null) {
return;
}
Client.getFacade().addTitleBook(data);
}
public String[] form_title() {
if (content_validate(val: title) == null) {
return null;
}
if (content_validate(val: ISBN) == null) {
return null;
}
if (content_validate(val: publisher) == null)
return null;
}
if (content_validate(val: author) == null) {
return null;
}
Search Results
Output
abase Process
GlassFish Server5
Library1_client1_SE (run)
100:1
INS

Client.java | Book_form.java

istory

```
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().clear
}
```

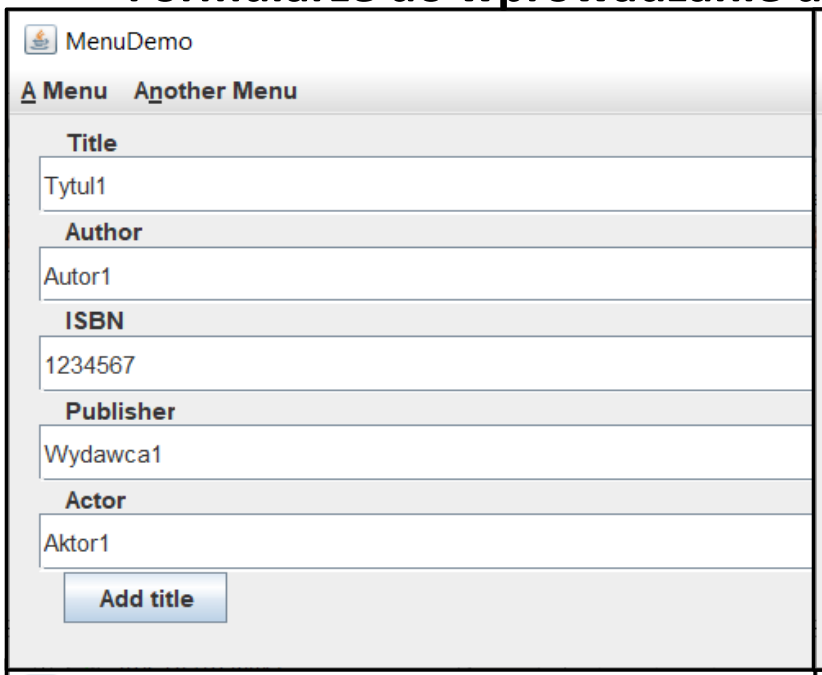
Wynik metody **getTiteBookModel** jest wykorzystany jako **model widoku JTable**

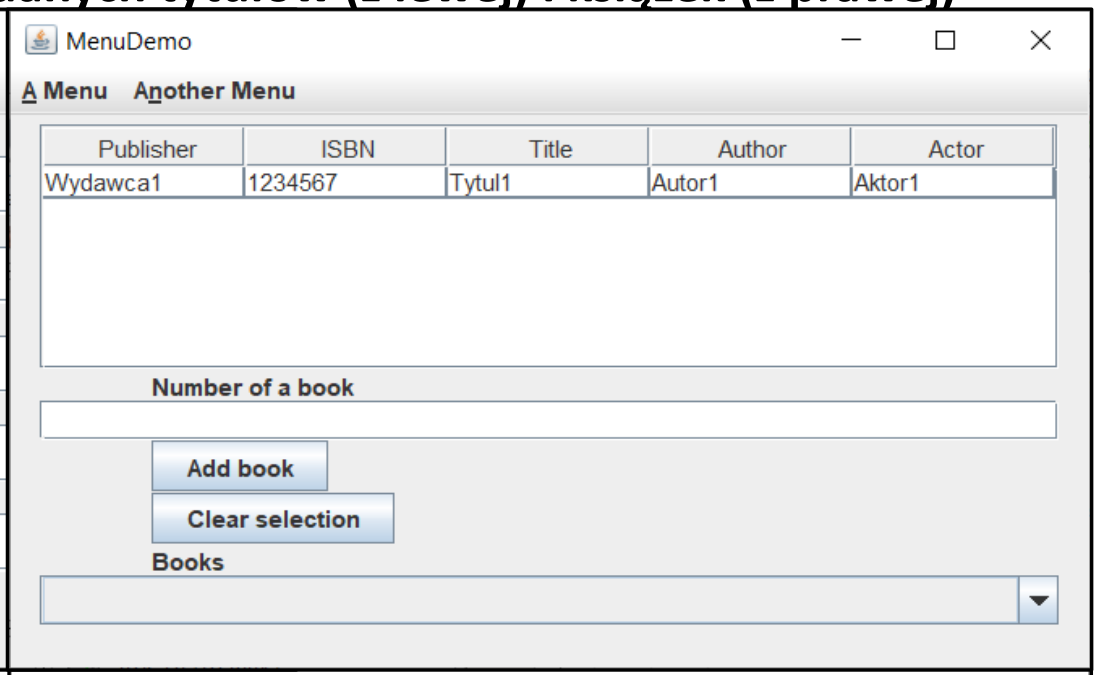
Udostępnianie logiki biznesowej w ***Warstwie klienta*** **(formularz do wprowadzania danych książki)** za pomocą metody **addBook** wzorca **Facade**,

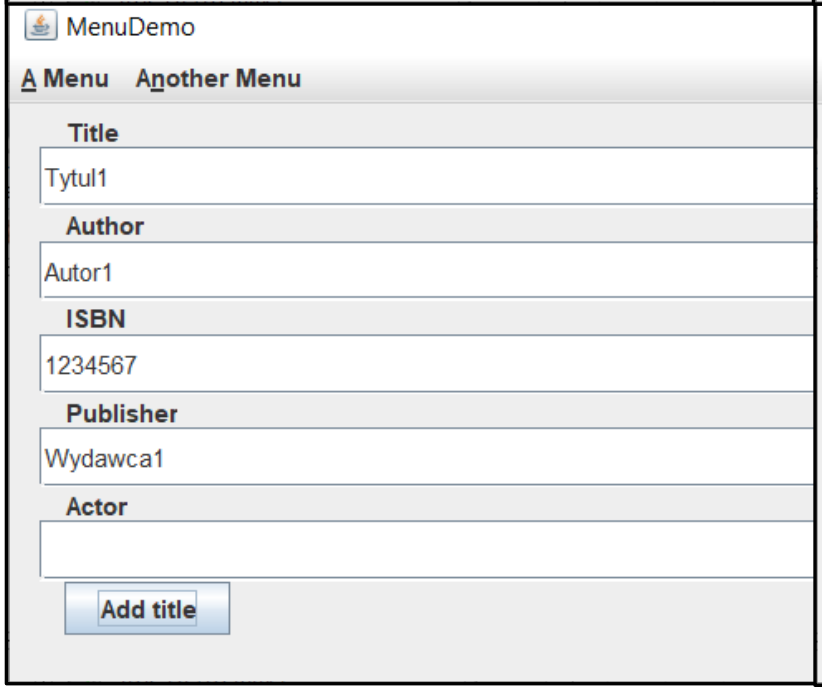
Wynik **help3** metody **addBook** jest wykorzystany jako **model widoku JComboBox**

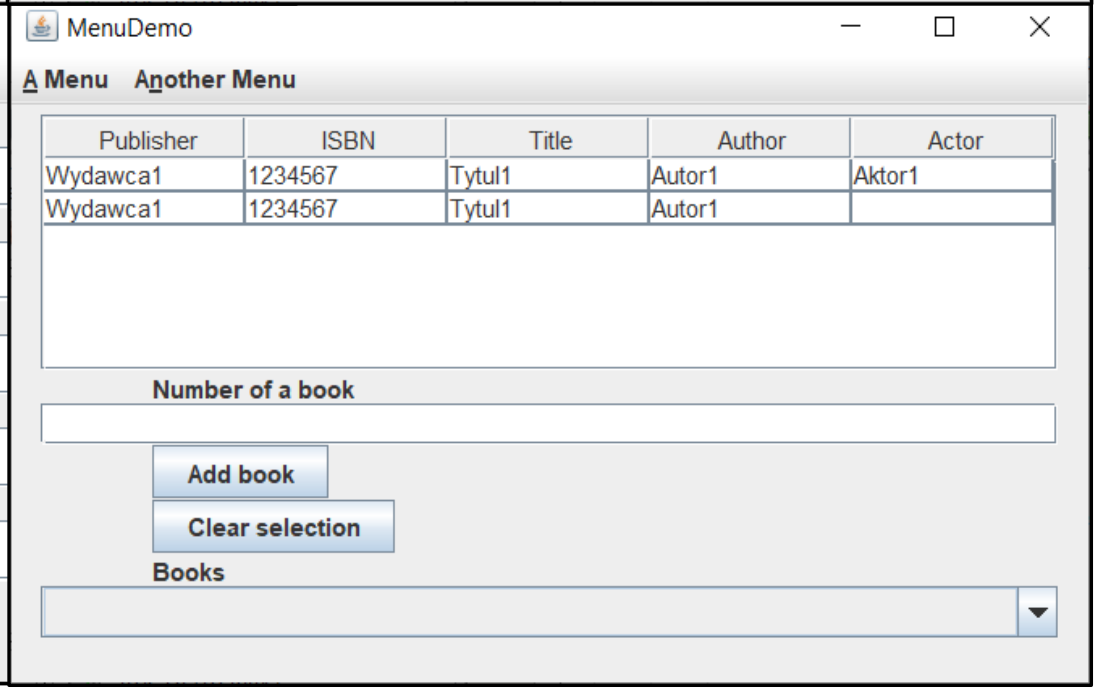
```
private void list_content(ArrayList<String> col, JComboBox list)
{    String s;
     list.removeAllItems();
     Iterator<String> iterator = col.iterator();
     while (iterator.hasNext()) {
        s = iterator.next();
        list.addItem(s);  }
  }
```

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

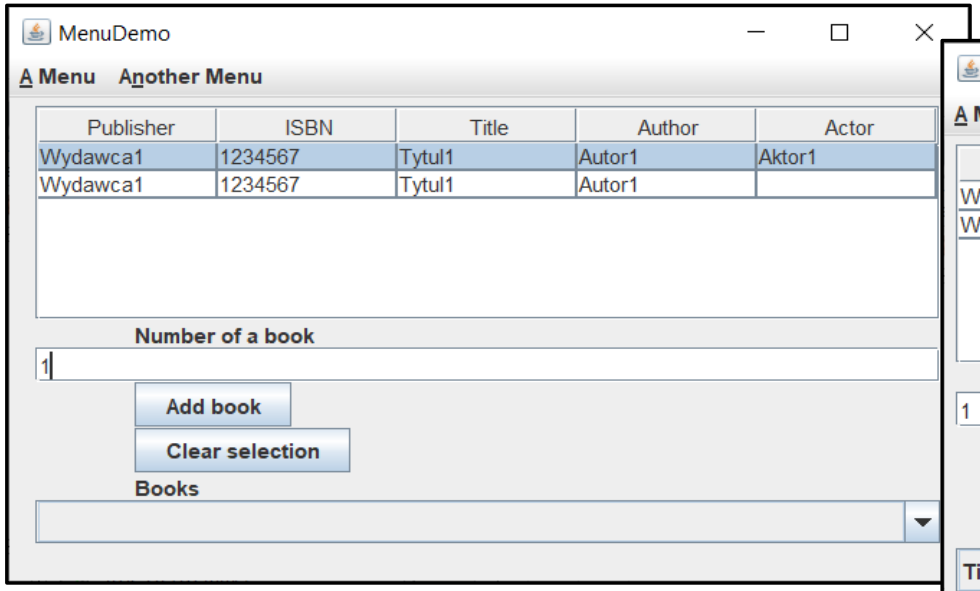


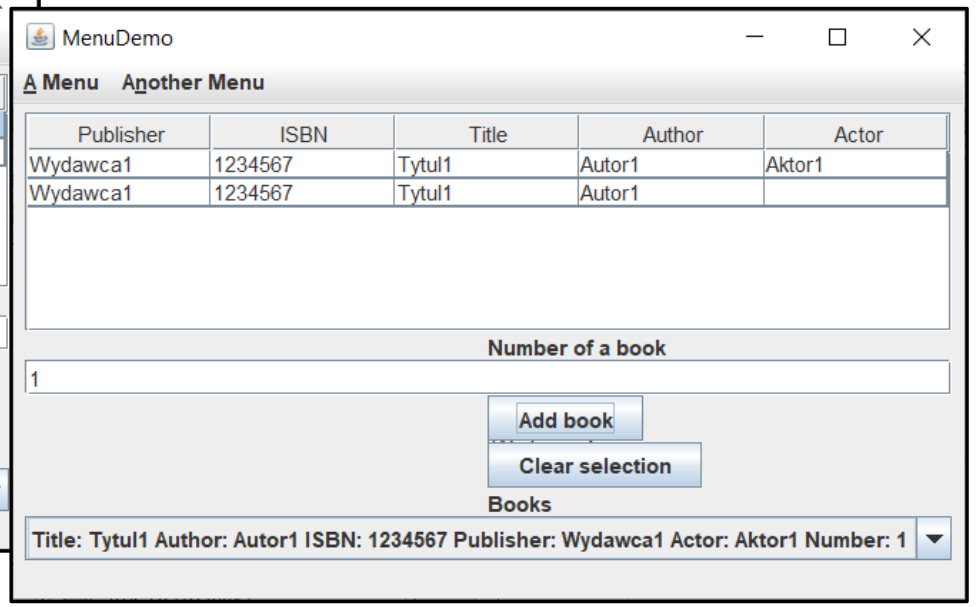


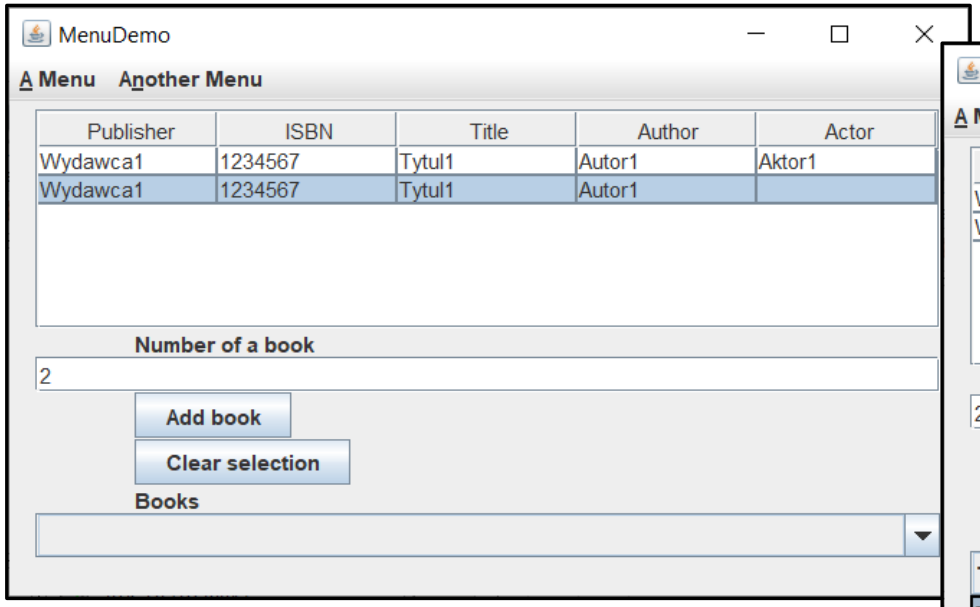


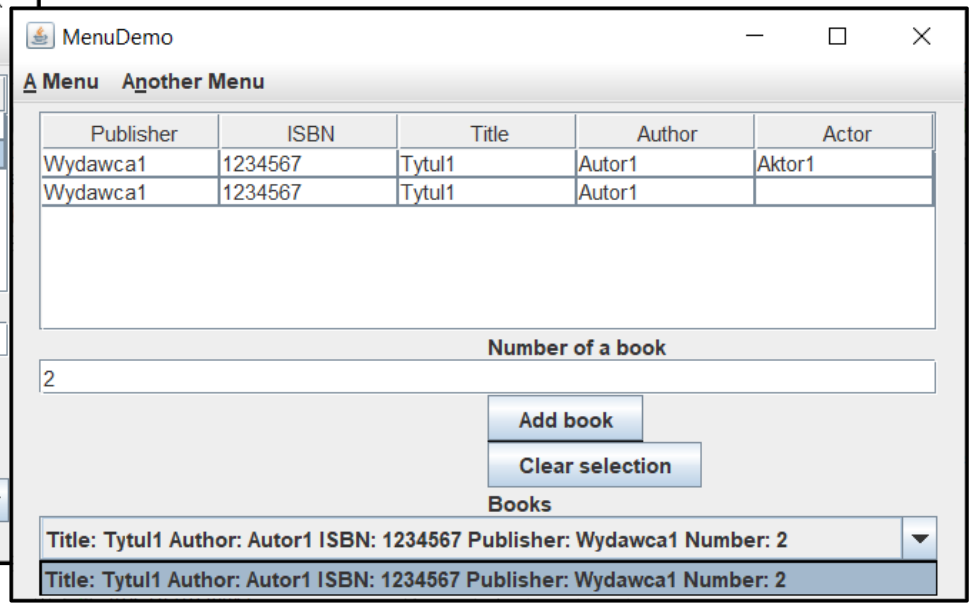


Wprowadzanie danych książek papierowych i nagranych (cd)









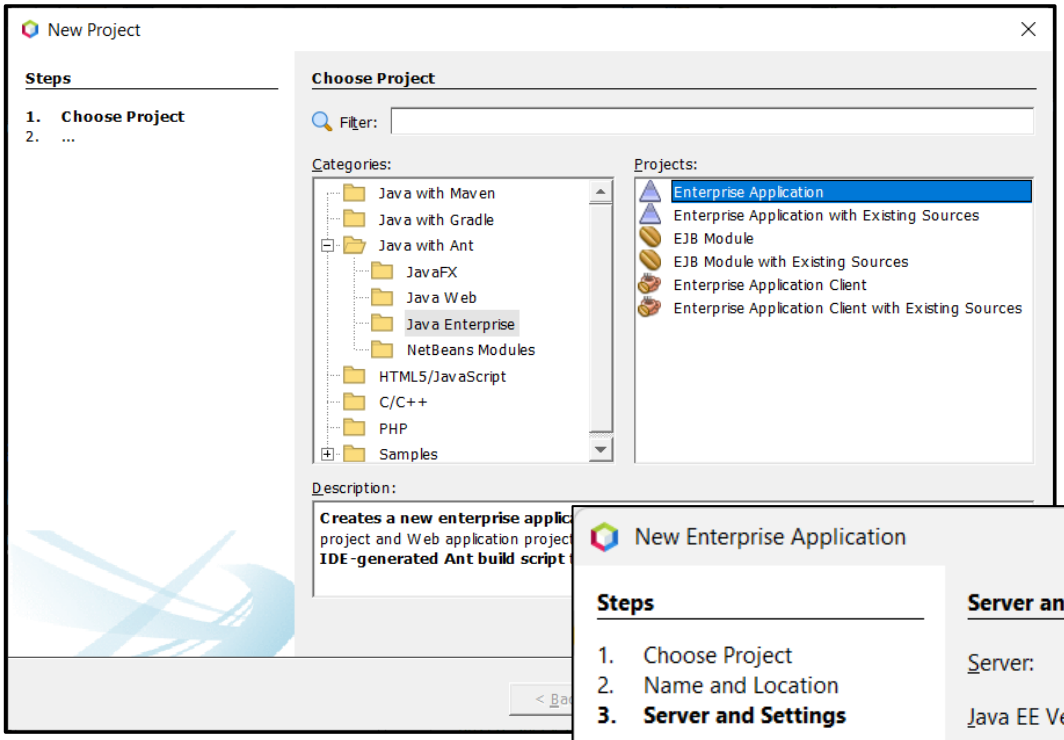
Zagadnienia

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

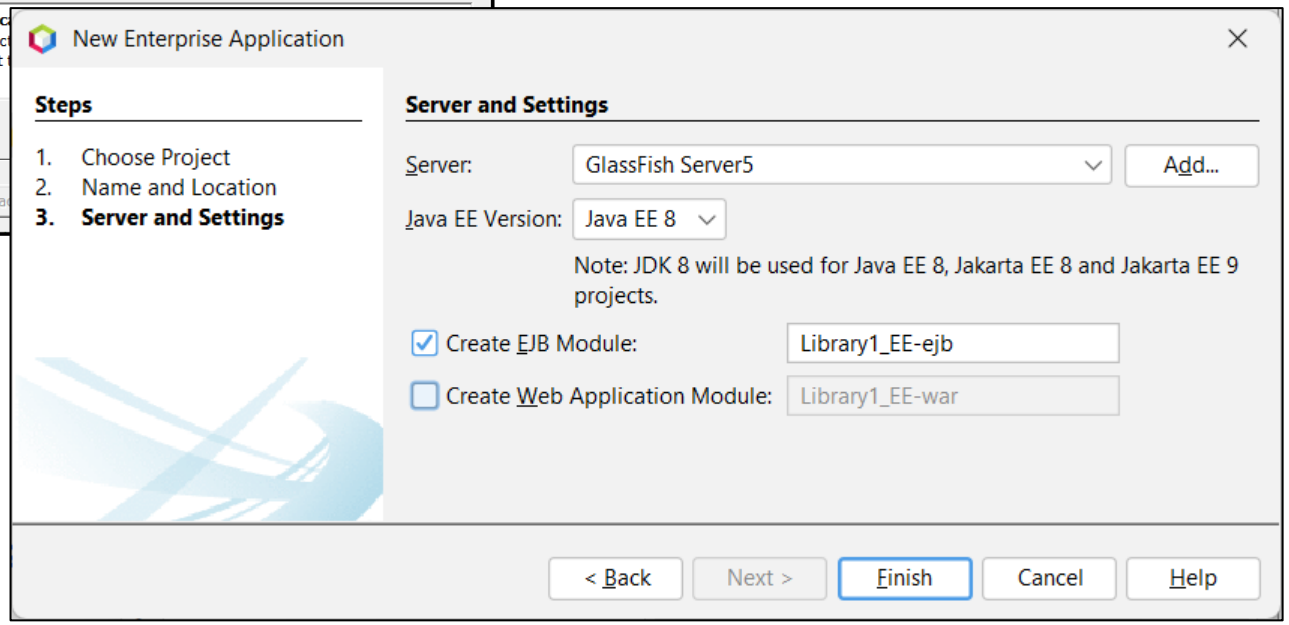
Architektura oprogramowania

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

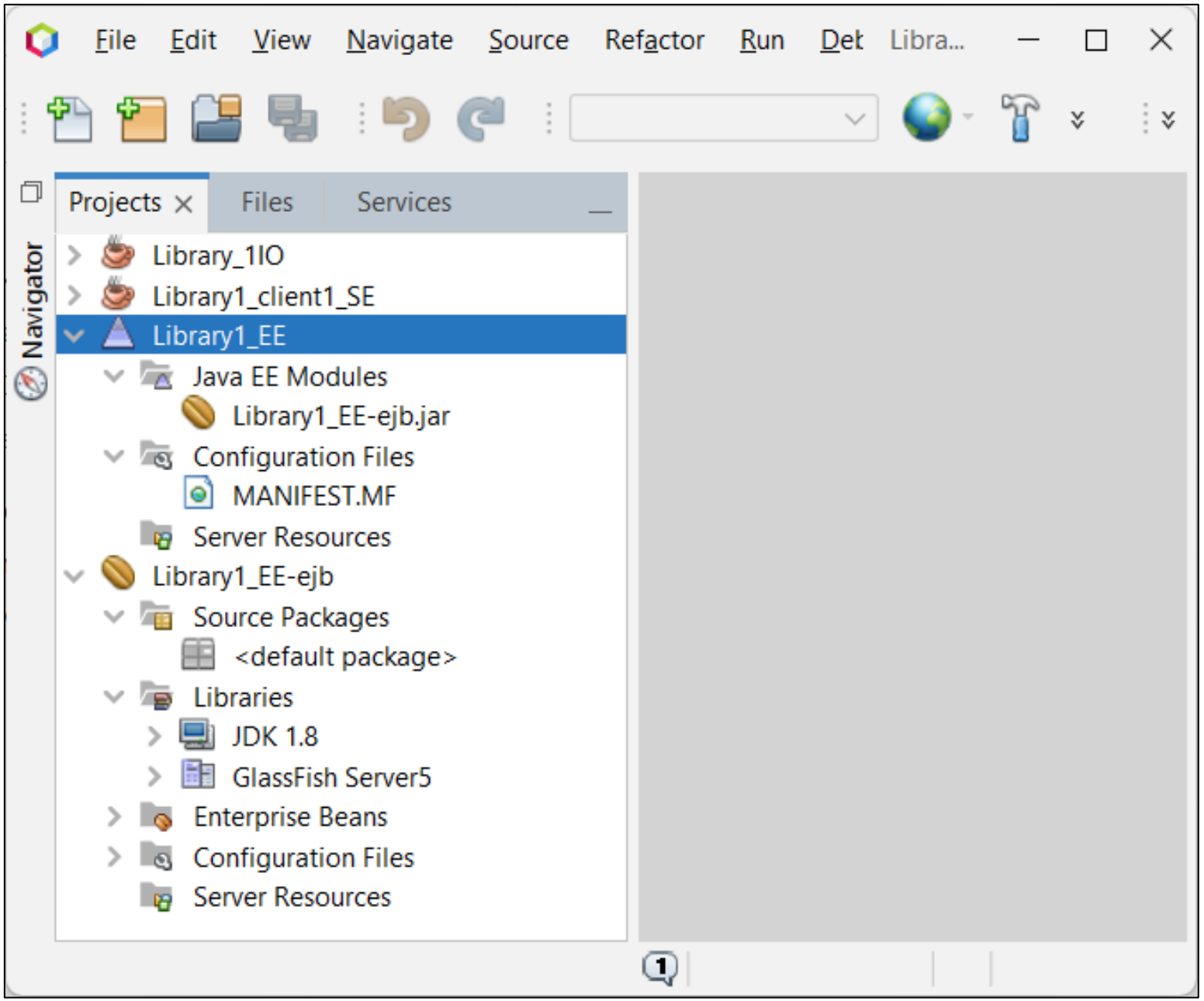


1. Tworzenie aplikacji typu **Enterprise Application**: **Library1_EE**

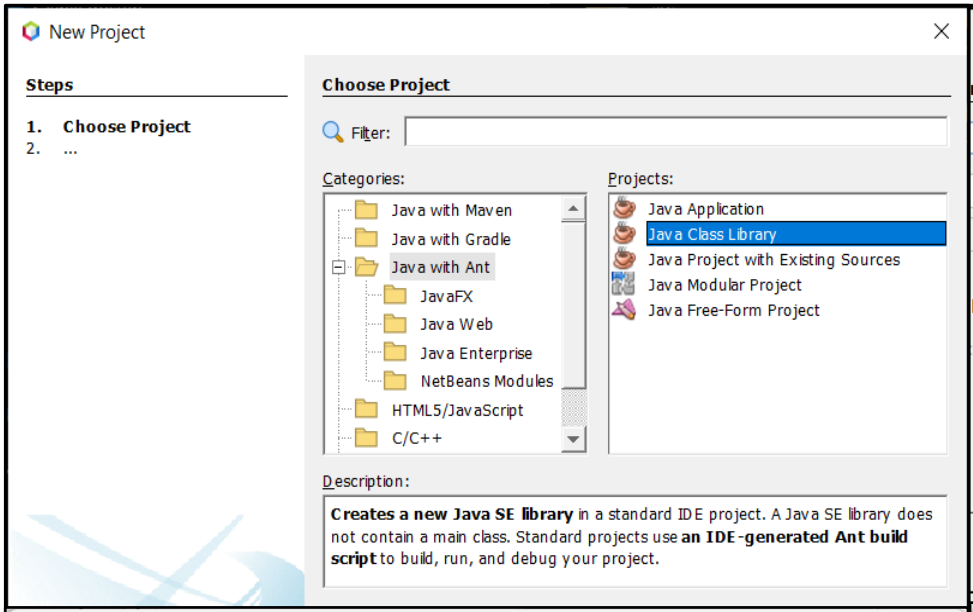


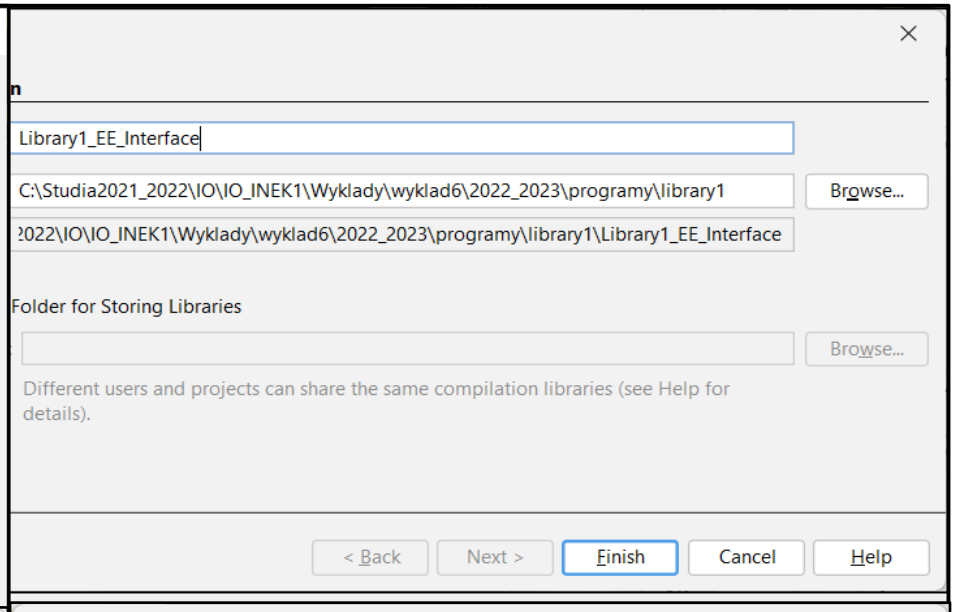
2. Dodanie modułu **EJB** w aplikacji **Enterprise Application**: **Library1_EE-ejb**

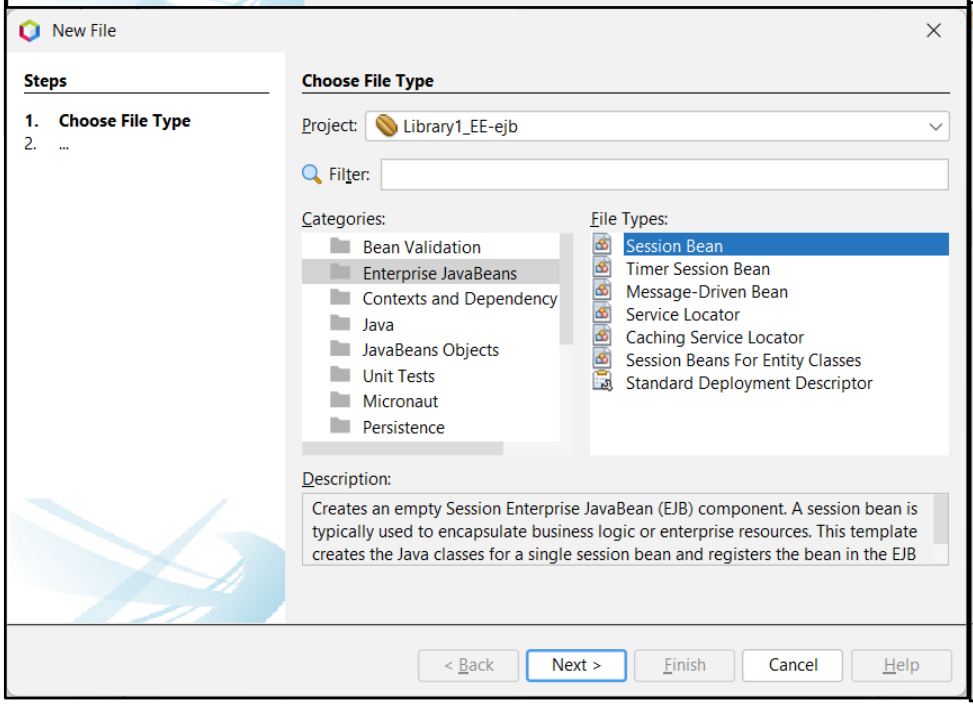
3. Projekt typu Enterprise Application

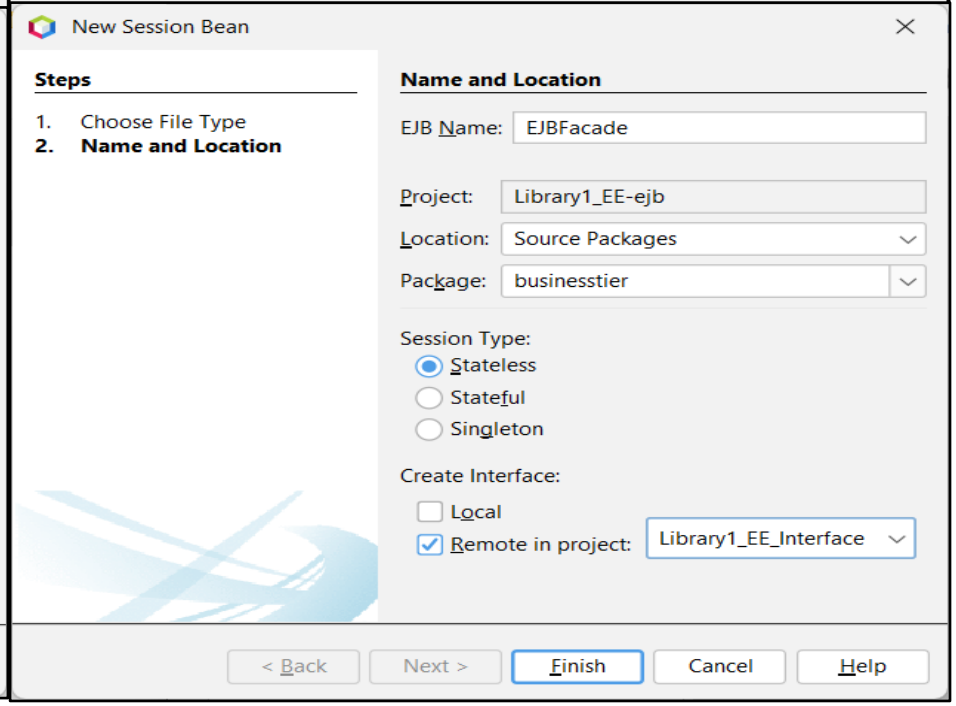


4. Wstawianie zdalnego komponentu EJB typu Session Bean do modułu EJB

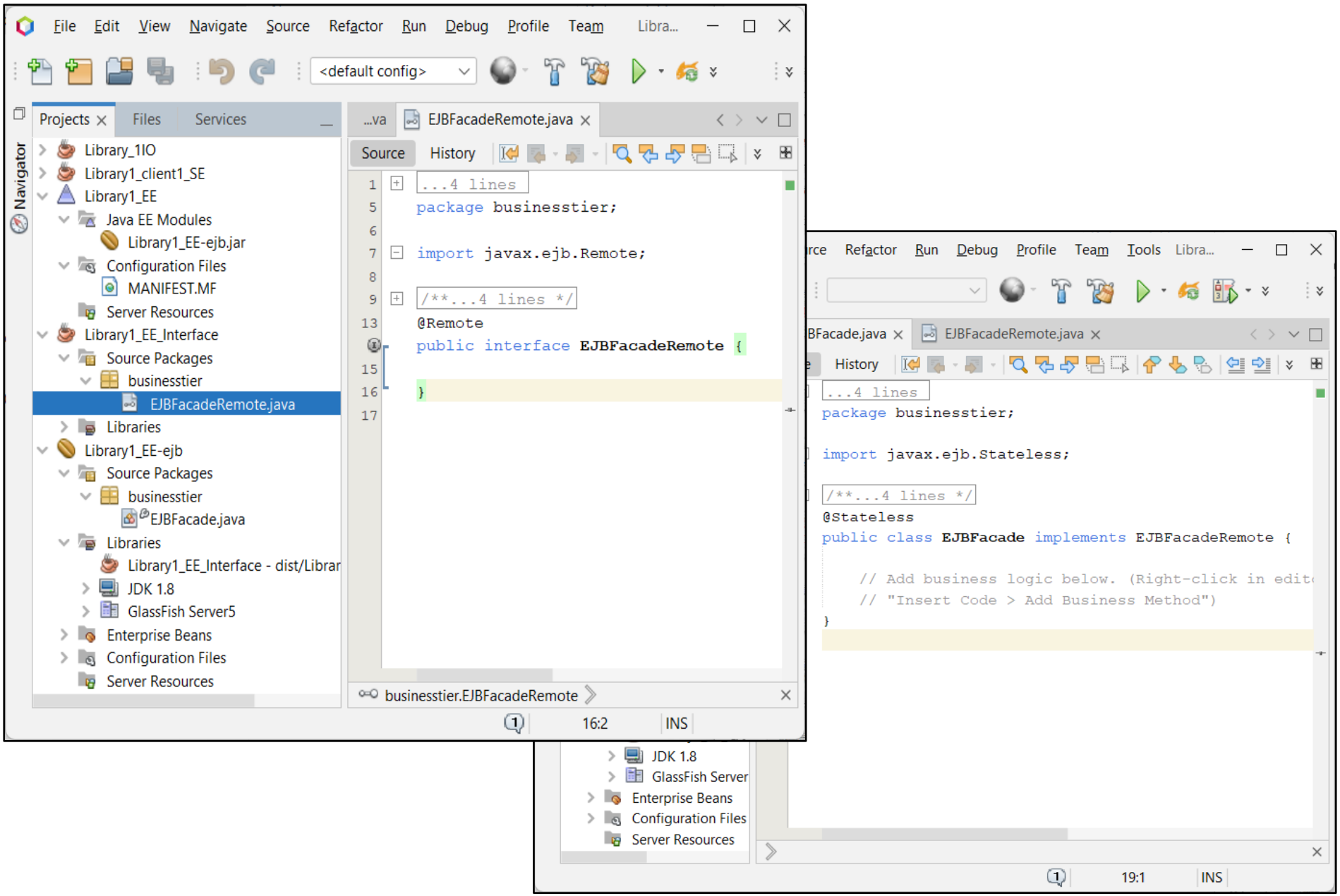




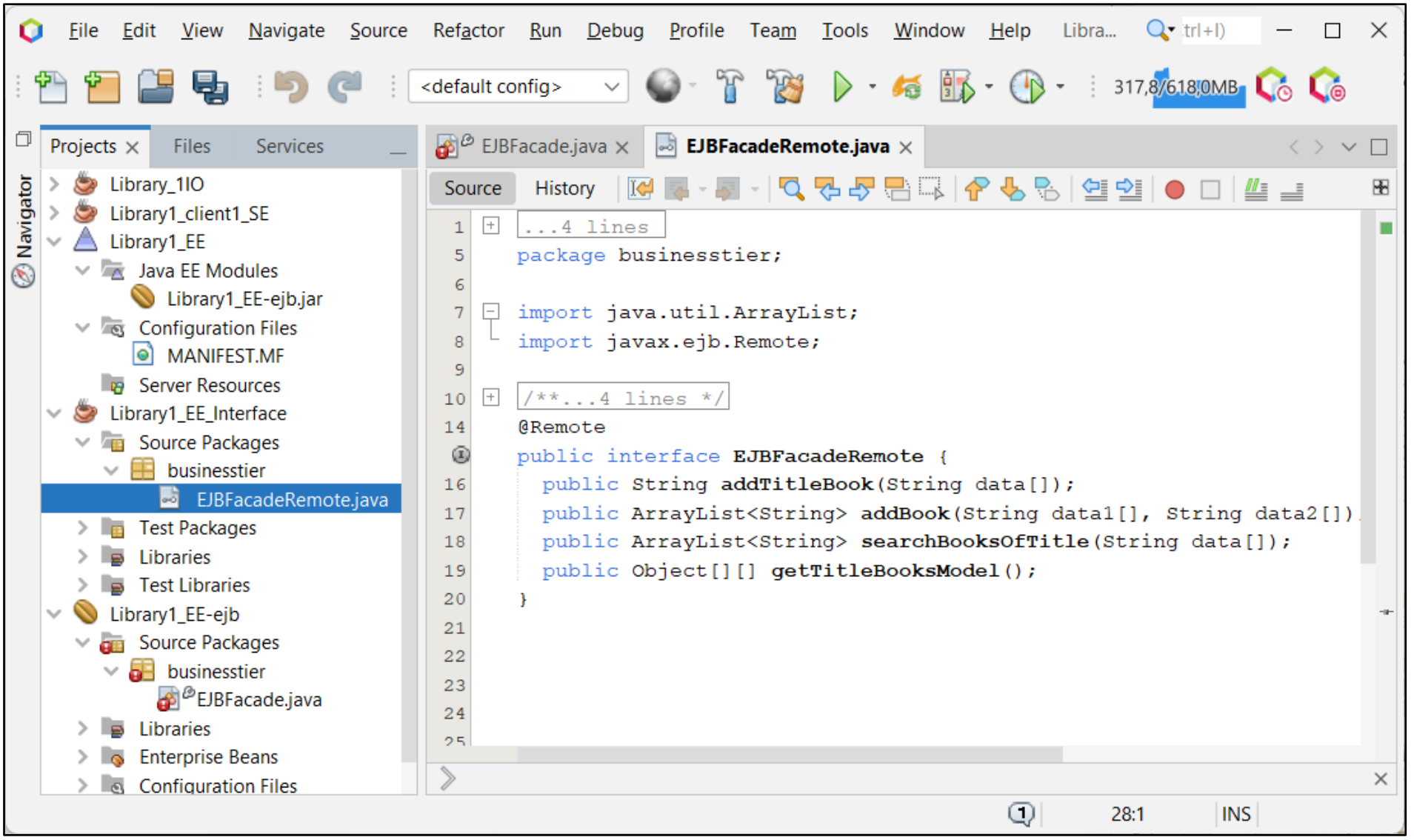




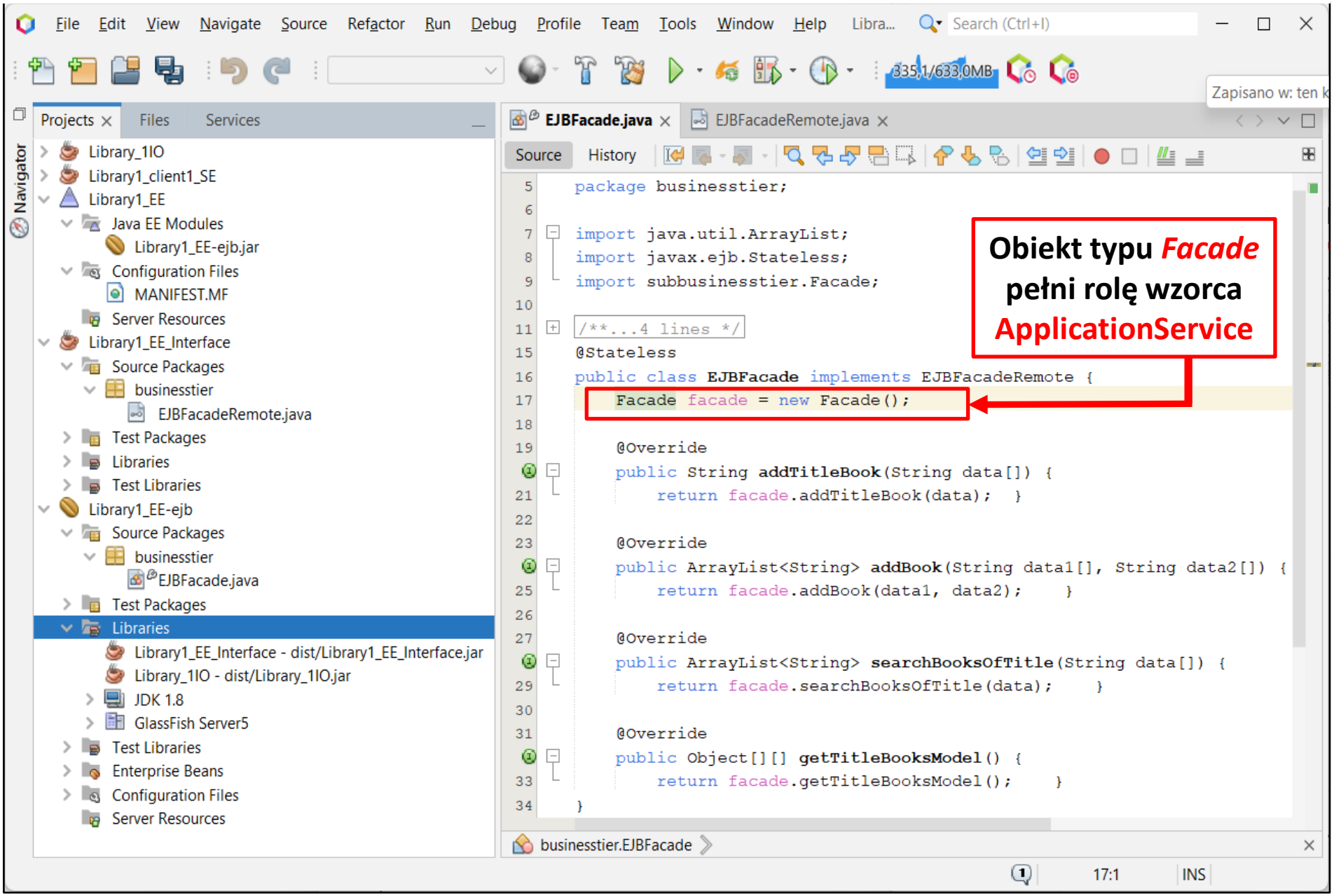
5. Wygenerowany: zdalny komponent Session Bean: EJBFacade w Library1_EE-ejb i jego interfejs EJBFacadeRemote w projekcie Library1_EE_Interface

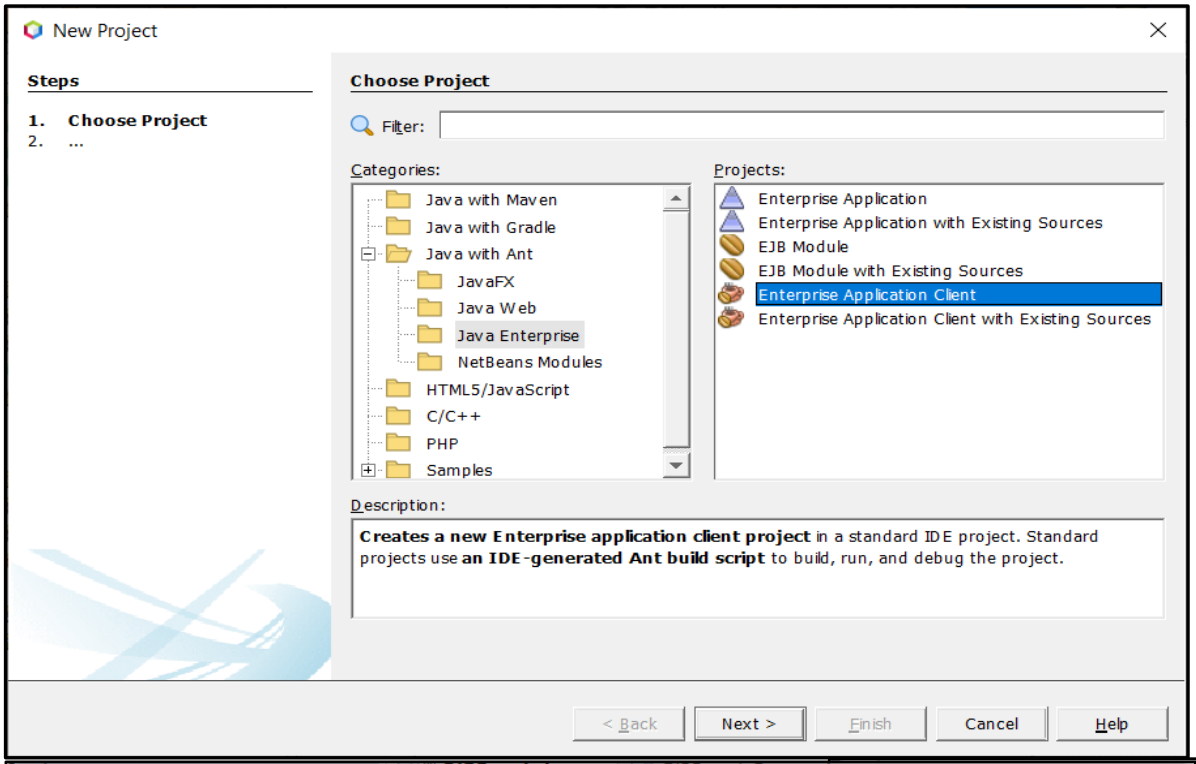


6. Deklaracja metod o zdalnym dostępie do metod logiki biznesowej klasy Facade w interfejsie komponentu typu Session Bean



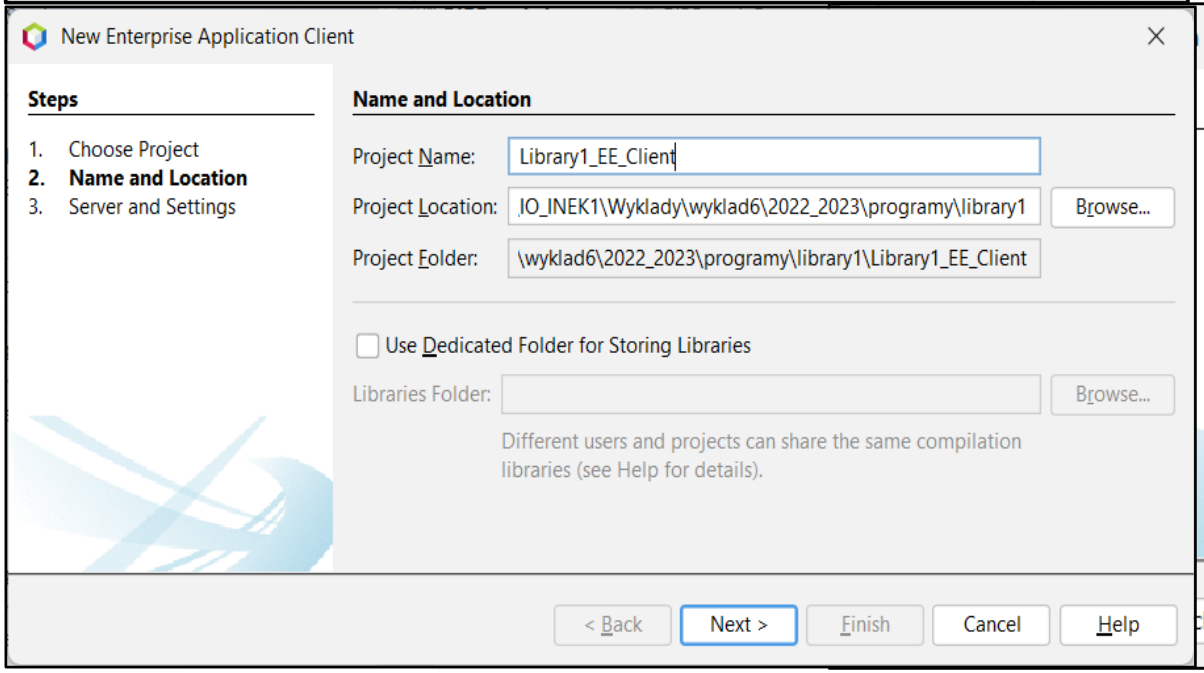
7. Definicja metod o zdalnym dostępie do metod logiki biznesowej klasy Facade (wzorzec ApplicationService) w komponencie Session Bean (wzorzec SessionFacade)

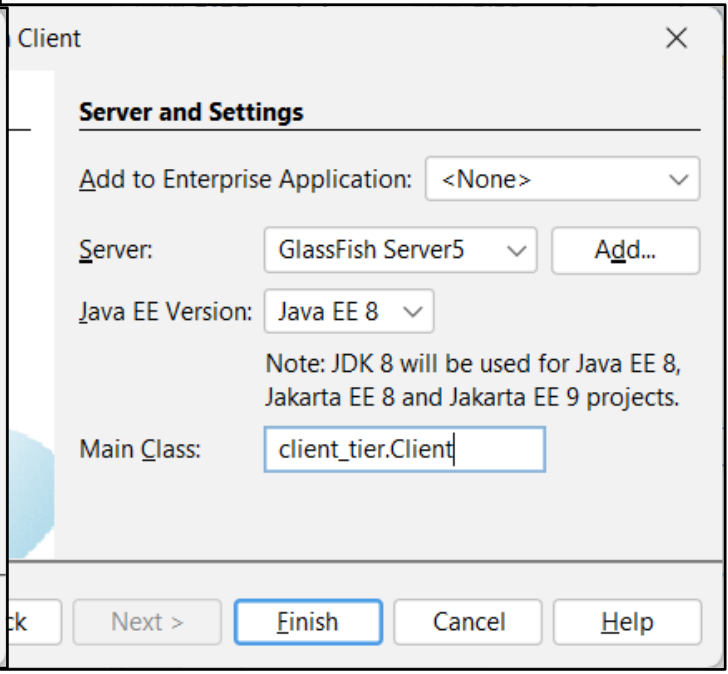


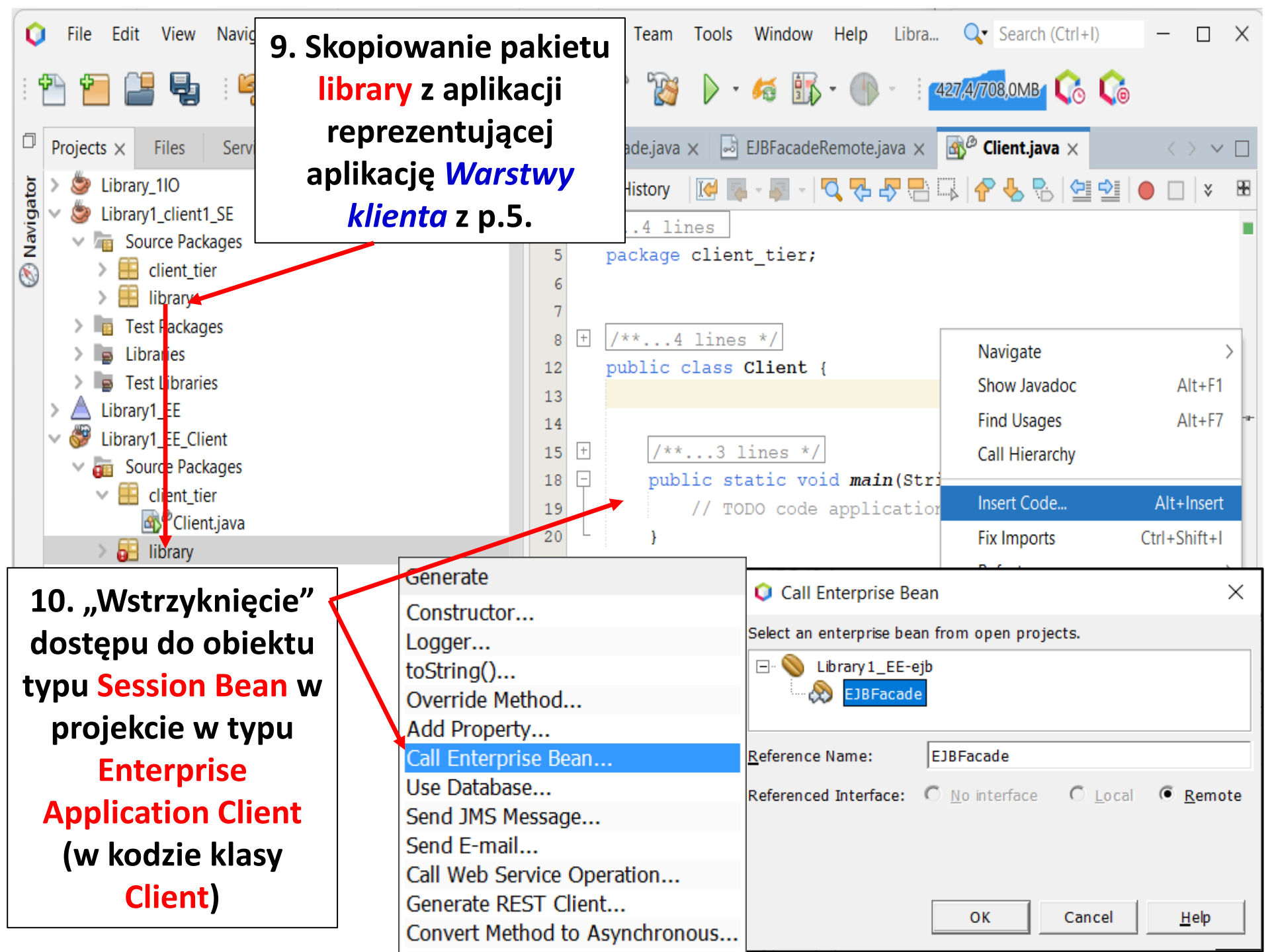


8. Wykonanie projektu typu Enterprise Application Client.

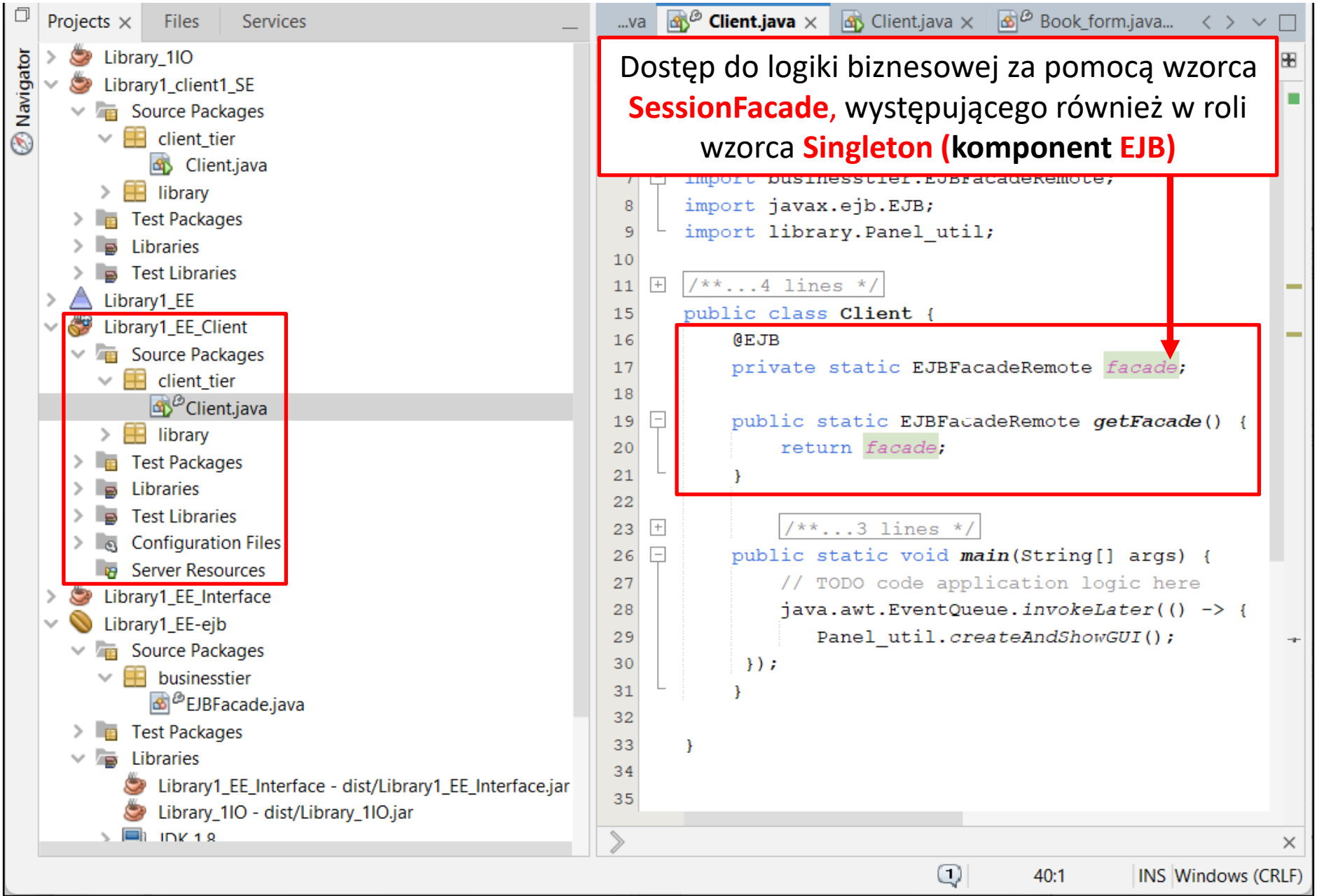
Kod tego projektu będzie wykonany w oparciu o pakiet library z projektu *Warstwy klienta* (aplikacja 2-warstwowa). Zdefiniowano domyślną klasę Client w pakiecie client_tier w celu uniknięcia modyfikacji kodu klas w pakiecie library.



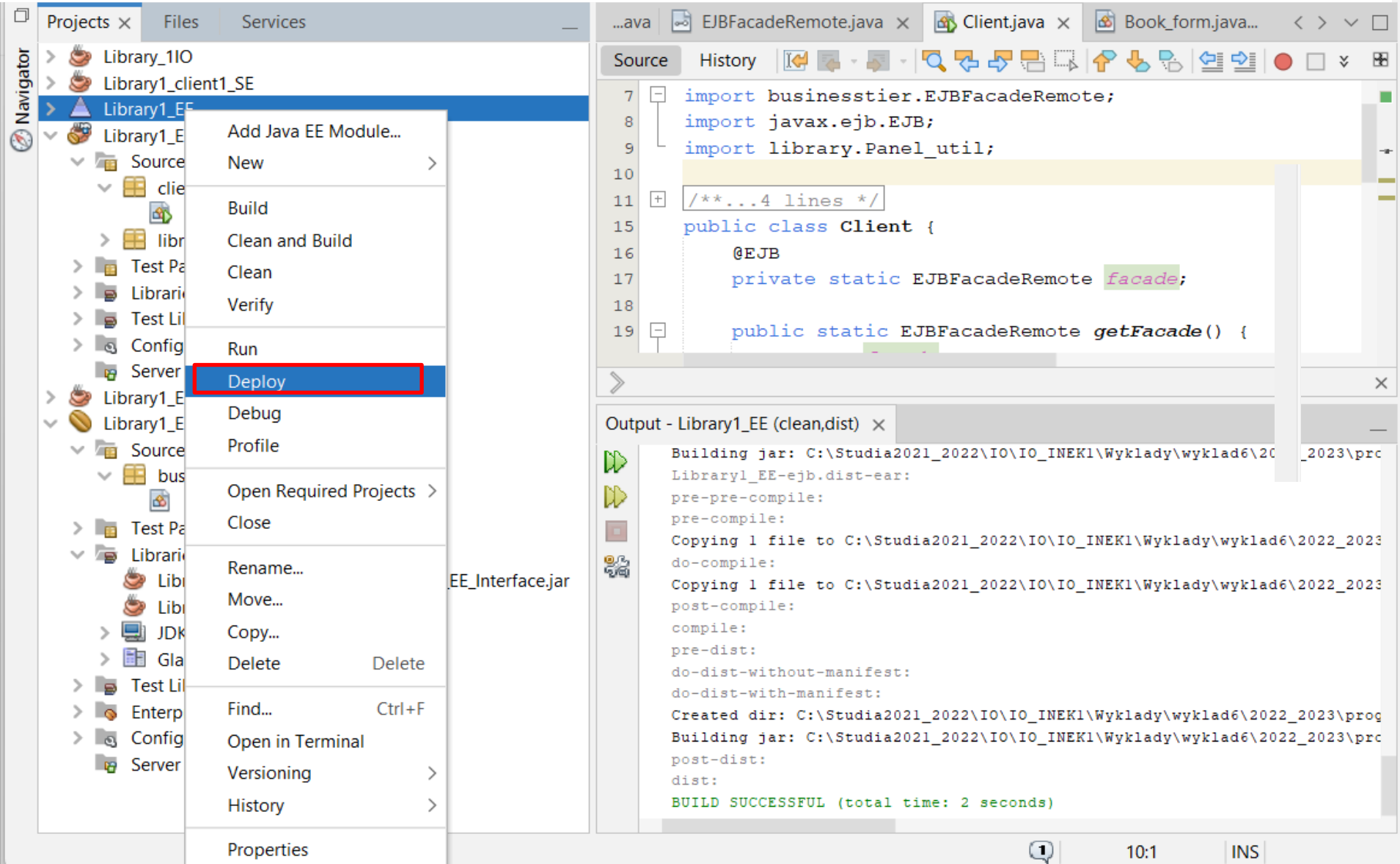



9. Skopiowanie pakietu library z aplikacji reprezentującej aplikację Warstwy klienta z p.5.
10. „Wstrzyknięcie” dostępu do obiektu typu Session Bean w projekcie w typu Enterprise Application Client (w kodzie klasy Client)
Projects
Files
Library_1IO
Library1_client1_SE
Source Packages
client_tier
library
Test Packages
Libraries
Test Libraries
Library1_EE
Library1_EE_Client
Source Packages
client_tier
Client.java
library
EJBFacadeRemote.java
Client.java
package client_tier;
public class Client {
Navigate
Show Javadoc Alt+F1
Find Usages Alt+F7
Call Hierarchy
Insert Code... Alt+Insert
Fix Imports Ctrl+Shift+I
Generate
Constructor...
Logger...
toString()...
Override Method...
Add Property...
Call Enterprise Bean...
Use Database...
Send JMS Message...
Send E-mail...
Call Web Service Operation...
Generate REST Client...
Convert Method to Asynchronous...
Call Enterprise Bean
Select an enterprise bean from open projects.
Library1_EE-ejb
EJBFacade
Reference Name: EJBFacade
Referenced Interface: No interface Local Remote
OK
Cancel
Help

11. Wykonana aplikacja Library1_EE_Client reprezentuje *Warstwę klienta* EE

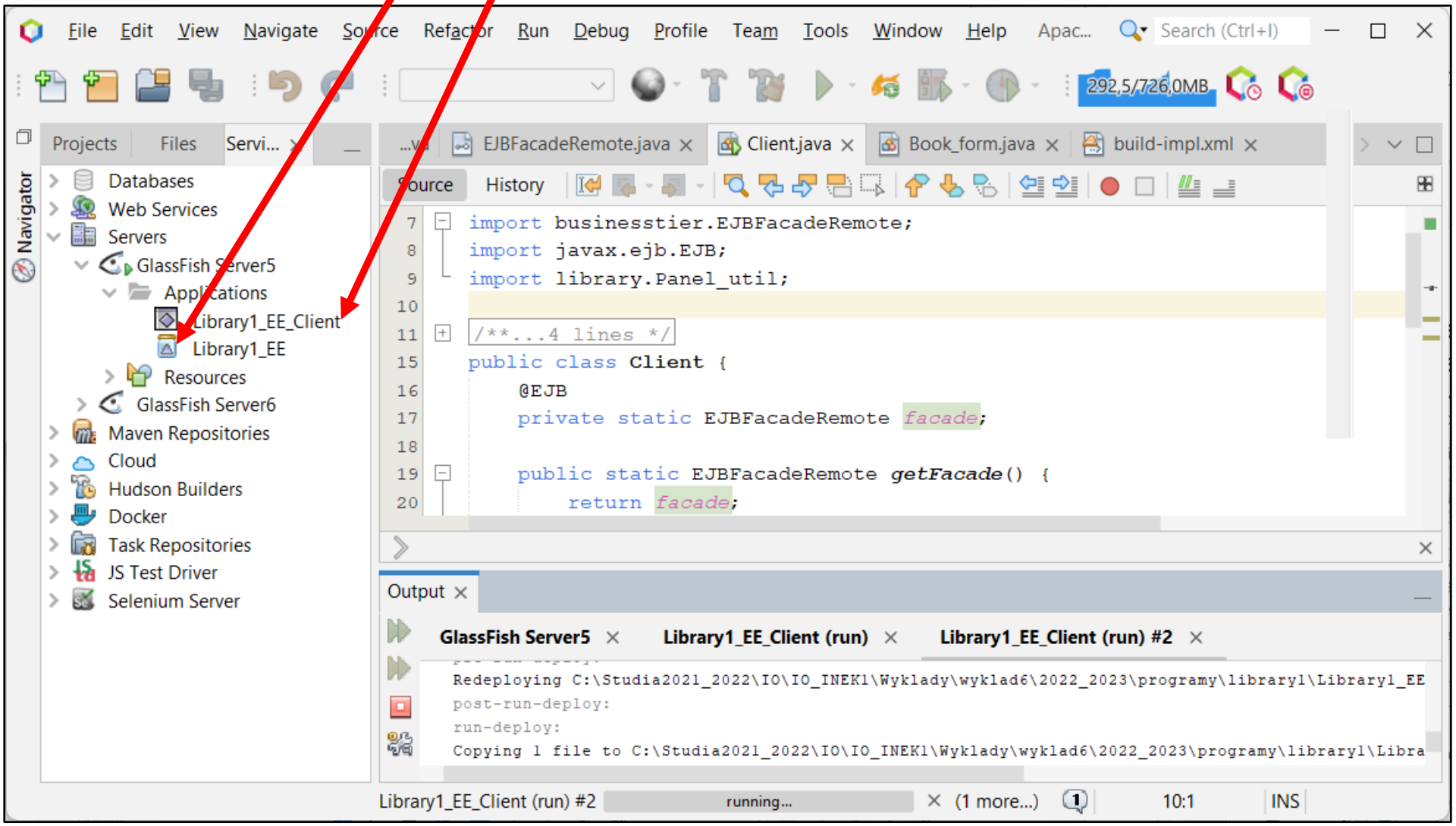


12. Uruchomienie desktopowej aplikacji wielowarstwowej typu EE: Deploy na aplikacji typu Enterprise Application (Library1_EE), a potem Run na projekcie typu Enterprise Application Client (Library1_EE_Client). Uruchomienie kolejnej aplikacji klienta wymaga jedynie powtórzenia operacji Run.

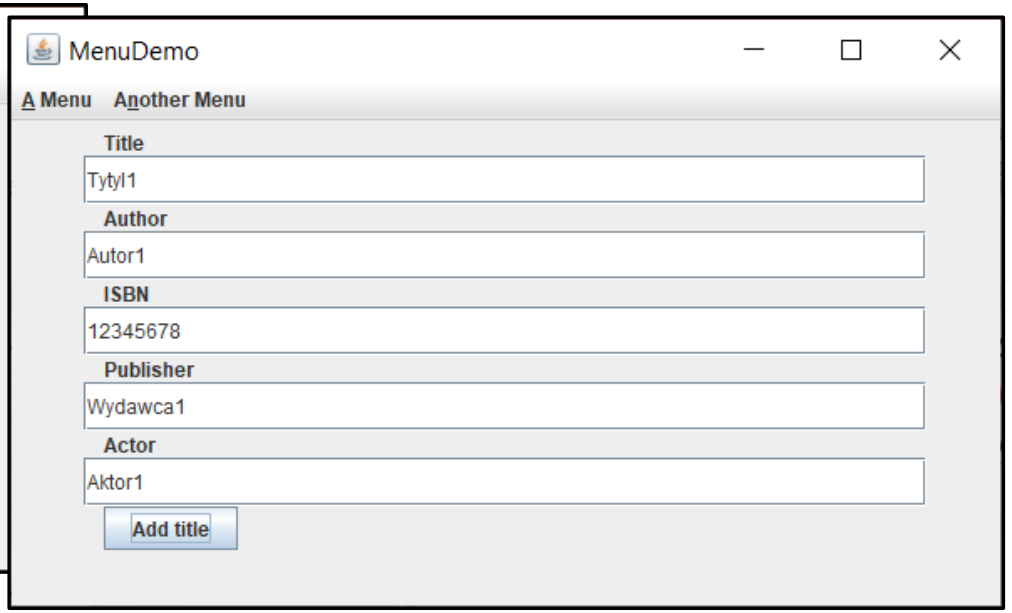


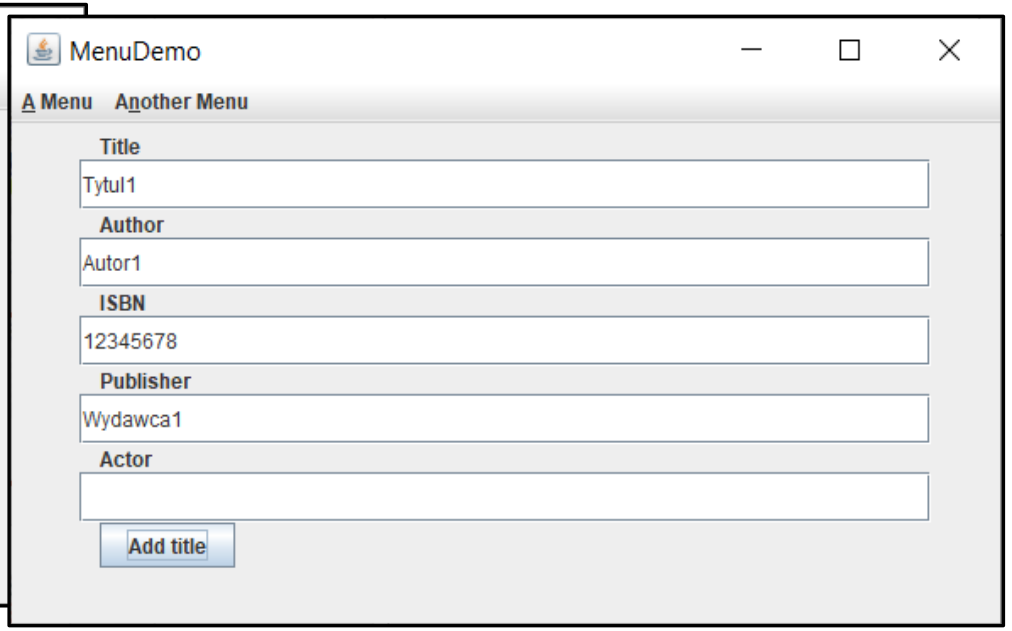
12(cd). Widok serwera aplikacji po operacjach:

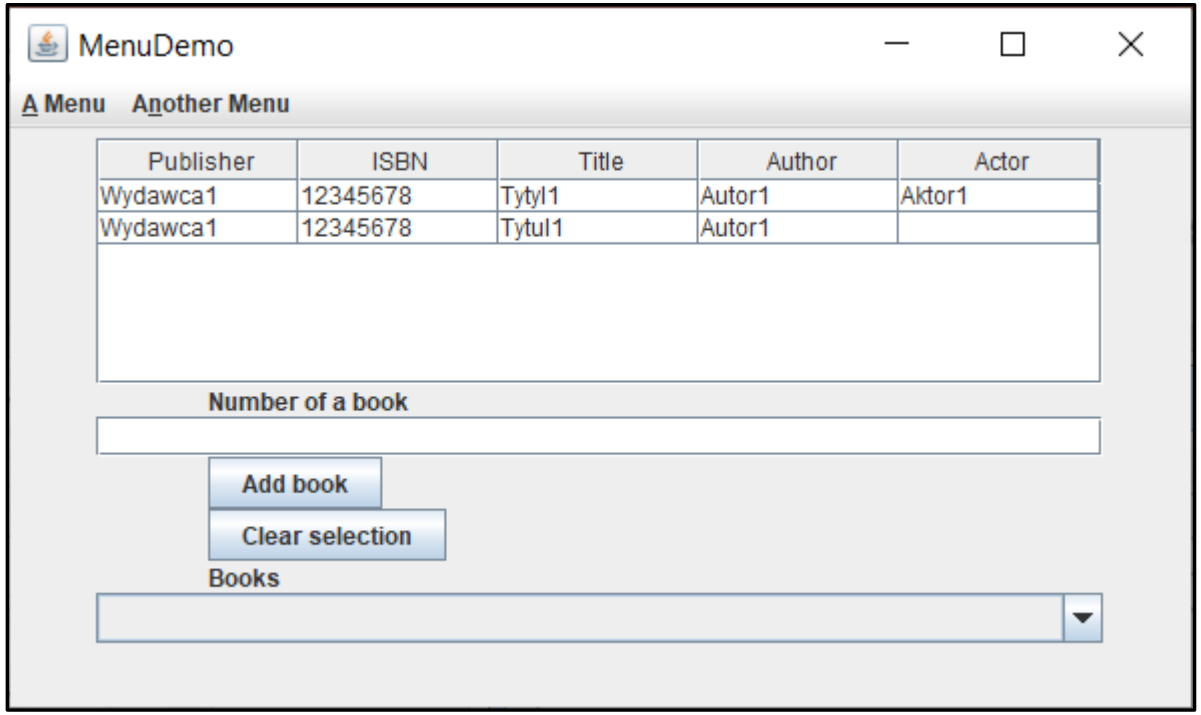
Deploy na aplikacji typu Enterprise Application (Library1_EE), a potem Run na projekcie typu Enterprise Application Client (Library1_EE_Client). Uruchomienie drugiej aplikacji Library1_EE_Client wykorzystuje ten sam kod na serwerze.

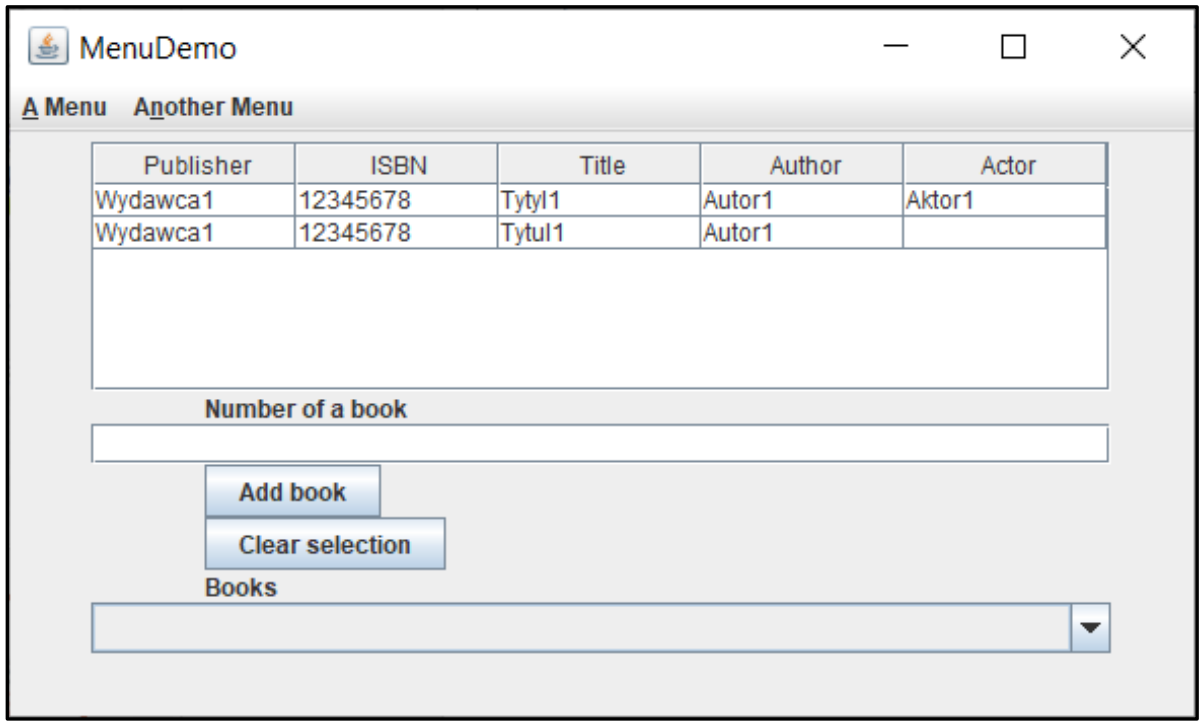


13. Formularze do wprowadzanie danych tytułów (z lewej) i książek (z prawej) – uruchomienie dwóch aplikacji w *Warstwie klienta* i wprowdzenie danych tytułów. Zostaną zapisane w *Warstwie biznesowej* w projekcie Library_1IO.

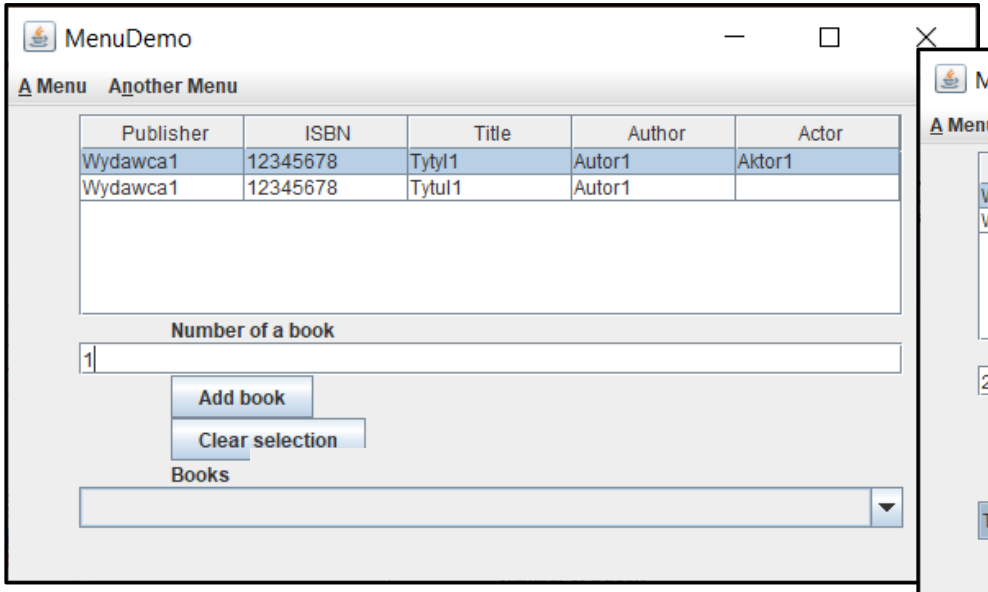


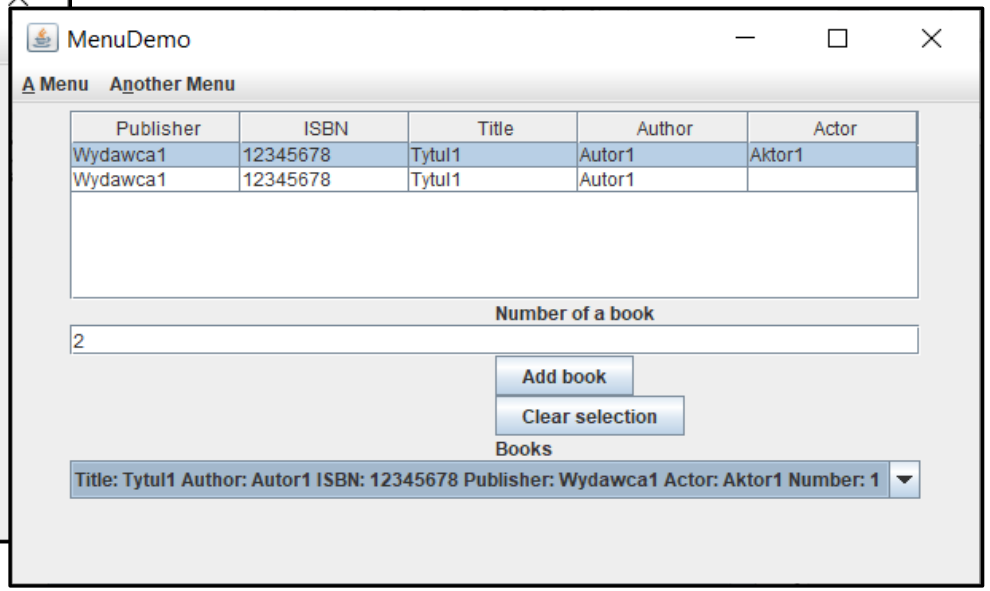




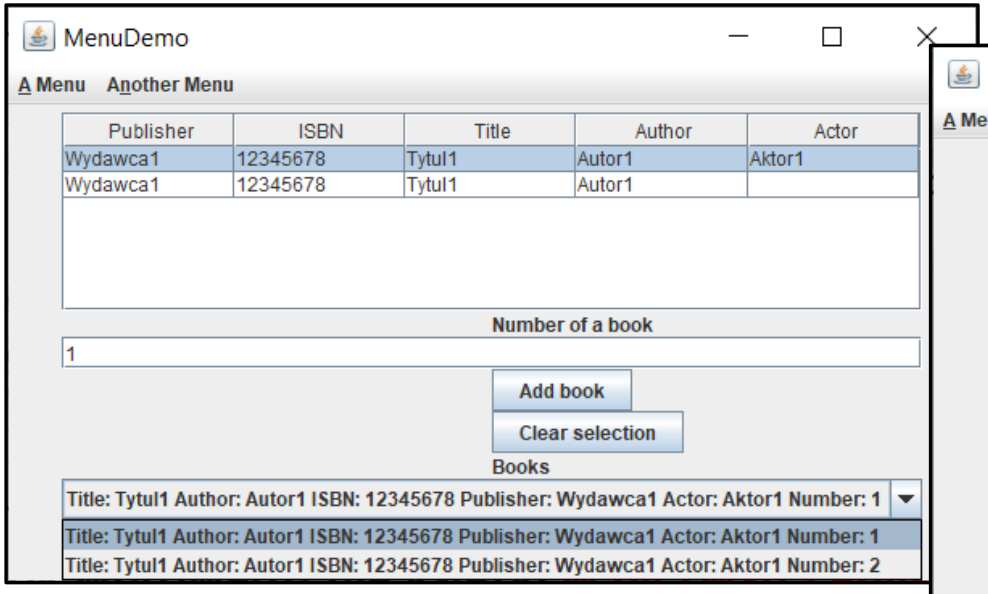


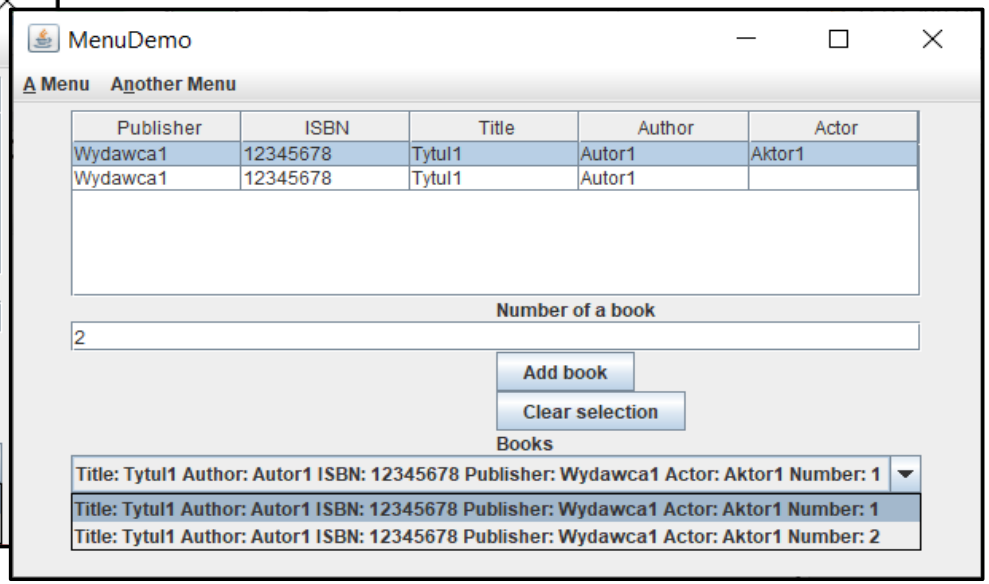
14. Widok stron do wprowadzania książek w obu uruchomionych aplikacjach klienckich





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



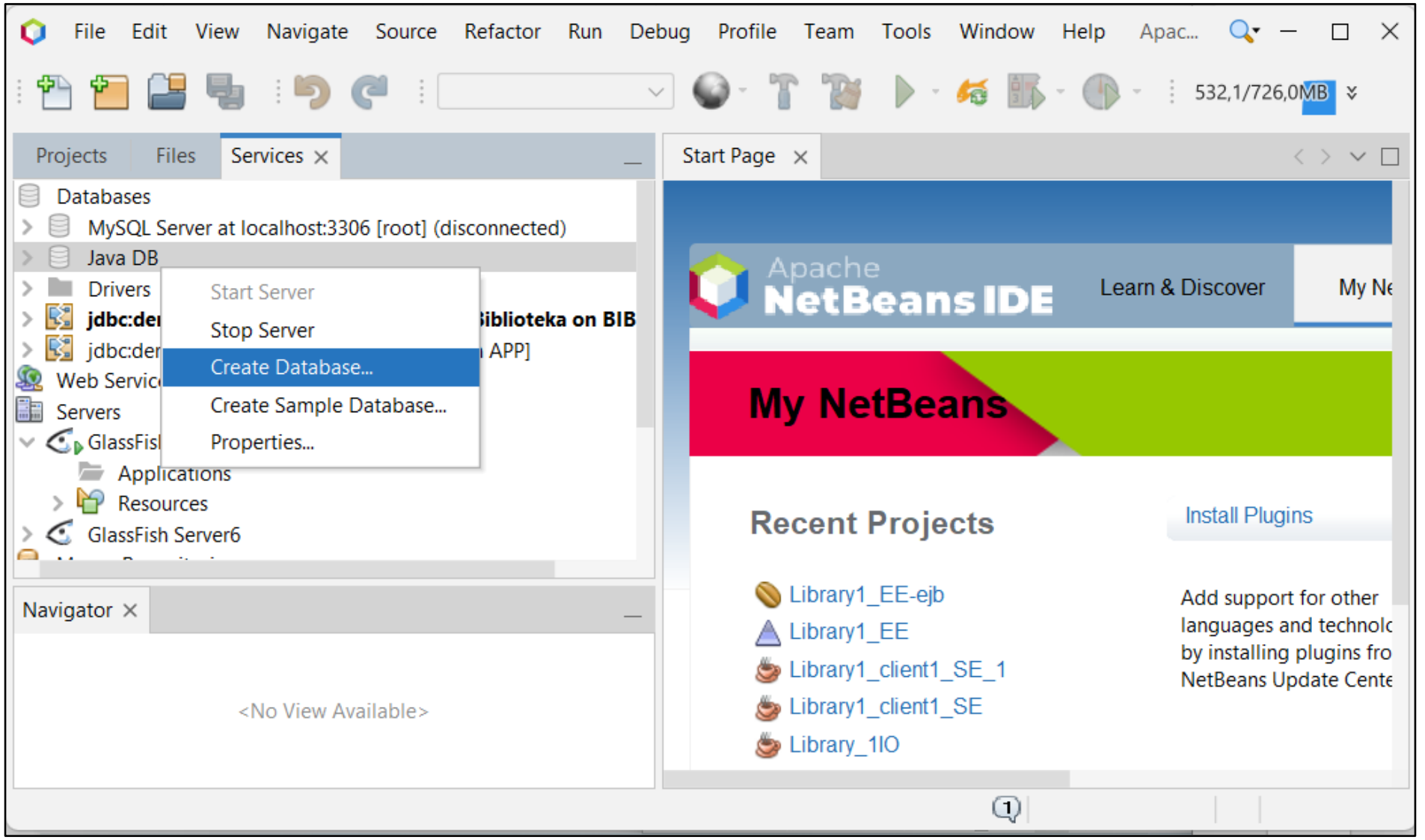


Zagadnienia

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.**

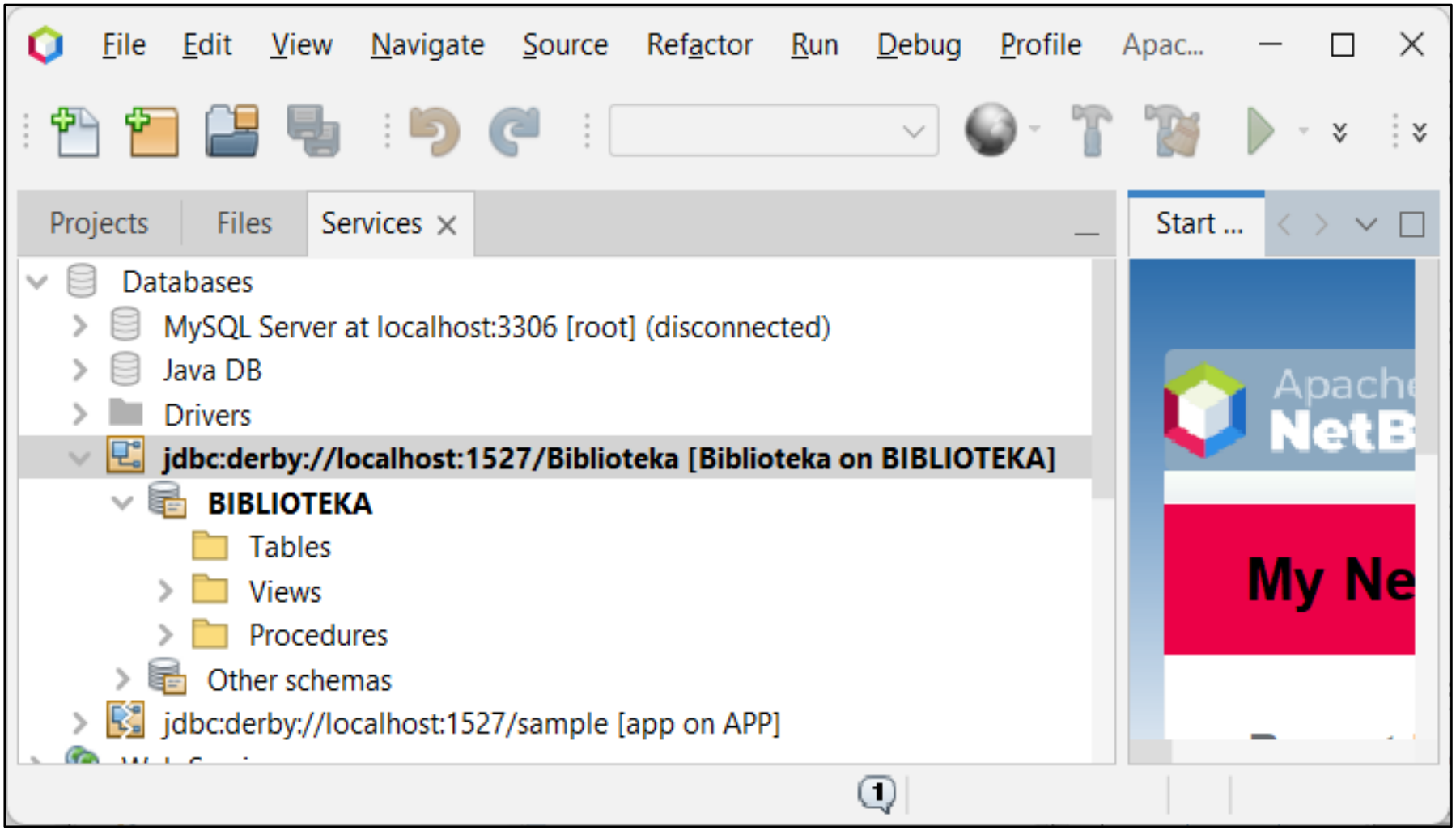
   ***Warstwa biznesowa*: komponenty typu EJB + obiekty POJO**
7. ***Warstwa zasobów* (EIS)- baza danych w systemie baz danych Apache Derby**

1. Zakładanie pustej bazy danych w systemie Apache Derby



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

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

Adaptery zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania

Warstwa zasobów
Baza danych Biblioteka typu Apache Derby

Zasoby, dane i usługi zewnętrzne

Zagadnienia

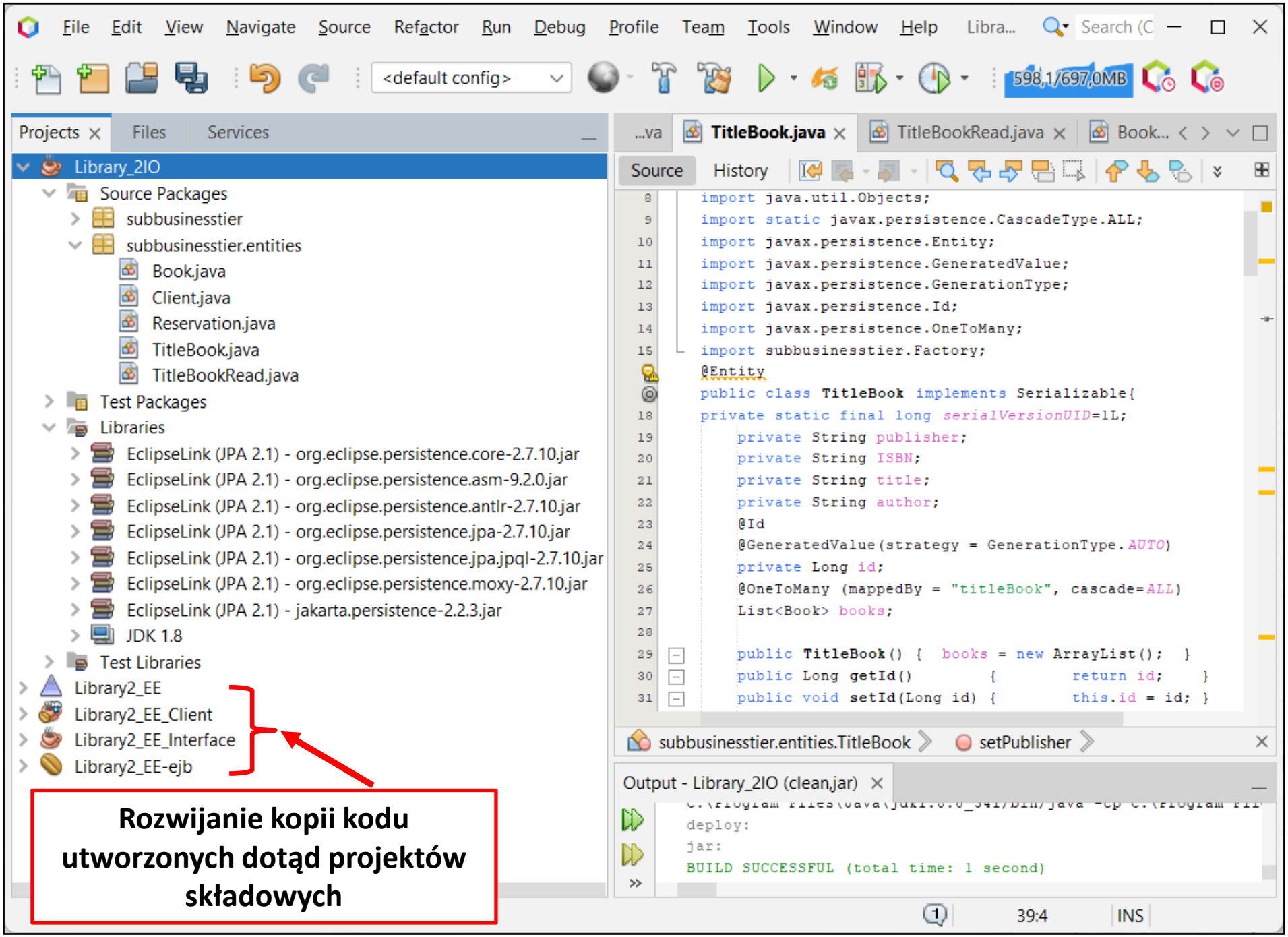
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego – przykłady architektury
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5.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

```java
@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 subbusinesstier.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 }

}
```

```java
@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 }
```


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
598,1/697,0MB
Projects
Files
Services
Library_2IO
Source Packages
subbusinesstier
subbusinesstier.entities
Book.java
Client.java
Reservation.java
TitleBook.java
TitleBookRead.java
Test Packages
Libraries
EclipseLink (JPA 2.1) - org.eclipse.persistence.core-2.7.10.jar
EclipseLink (JPA 2.1) - org.eclipse.persistence.asm-9.2.0.jar
EclipseLink (JPA 2.1) - org.eclipse.persistence.antlr-2.7.10.jar
EclipseLink (JPA 2.1) - org.eclipse.persistence.jpa-2.7.10.jar
EclipseLink (JPA 2.1) - org.eclipse.persistence.jpa.jpql-2.7.10.jar
EclipseLink (JPA 2.1) - org.eclipse.persistence.moxy-2.7.10.jar
EclipseLink (JPA 2.1) - jakarta.persistence-2.2.3.jar
JDK 1.8
Test Libraries
Library2_EE
Library2_EE_Client
Library2_EE_Interface
Library2_EE-ejb
TitleBook.java
TitleBookRead.java
Source
History
import java.util.Objects;
import static javax.persistence.CascadeType.ALL;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import subbusinesstier.Factory;
@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;
@OneToMany (mappedBy = "titleBook", cascade=ALL)
List<Book> books;
public TitleBook() { books = new ArrayList(); }
public Long getId() { return id; }
public void setId(Long id) { this.id = id; }
subbusinesstier.entities.TitleBook
setPublisher
Output - Library_2IO (clean,jar)
deploy:
jar:
BUILD SUCCESSFUL (total time: 1 second)
39:4
INS
Rozwijanie kopii kodu utworzonych dotąd projektów składowych

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.)

Warstwa klienta
Library2_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_2IO, Library2_EE_Interface
Library2_EE, Library2_EE-ejb(businesstier)

Logika biznesowa, transakcje, dane i usługi

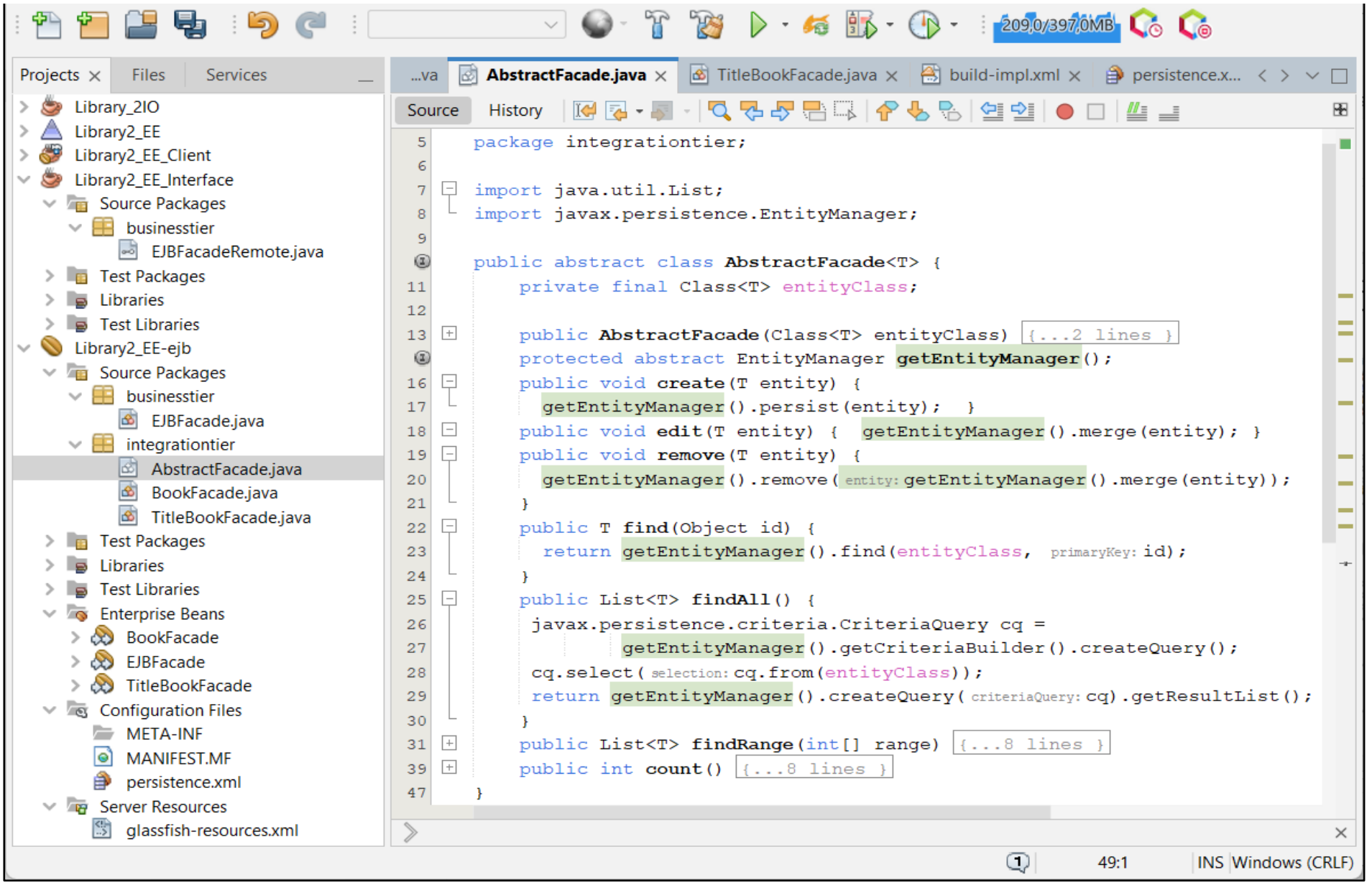
Warstwa integracji
Library2_EE-ejb (integrationtier)

Adaptery zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania

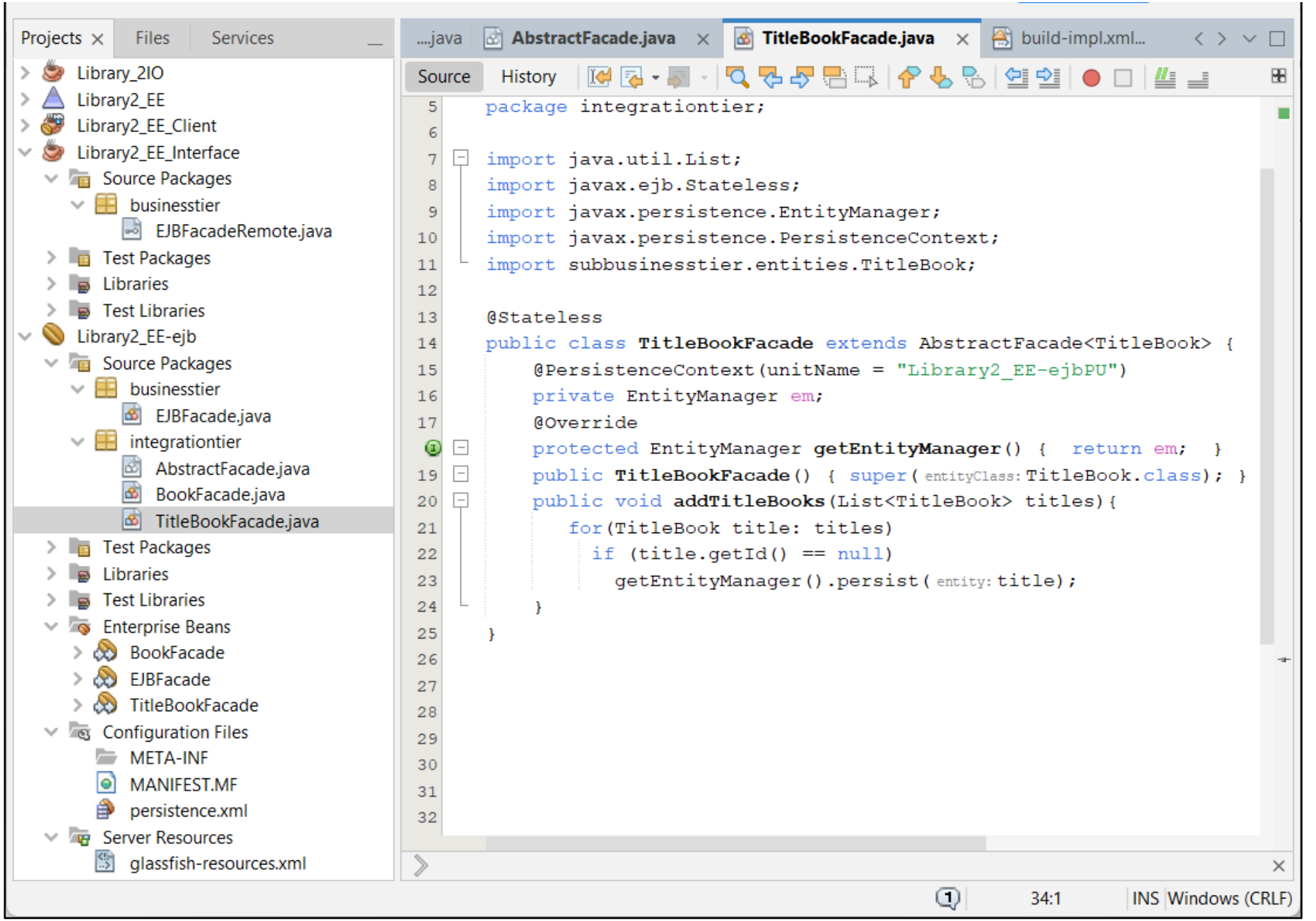
Warstwa zasobów
Baza danych Biblioteka typu Apache Derby

Zasoby, dane i usługi zewnętrzne

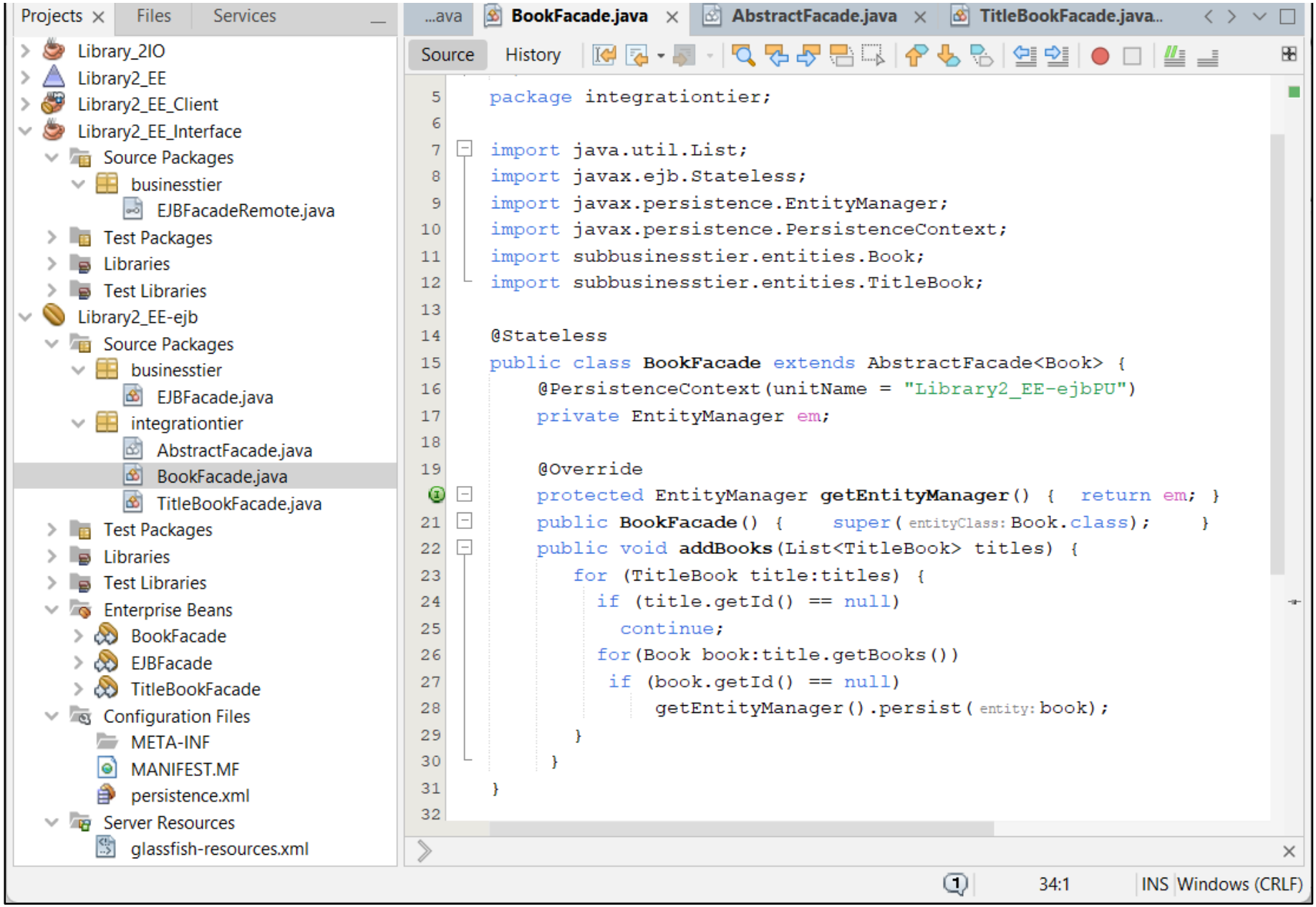
1. Klasa abstrakcyjna generyczna AbstractFacade z definicją uniwersalnych metod obsługujących transakcje JPA - parametr T może być zastąpiony każdą z klas typu Entity



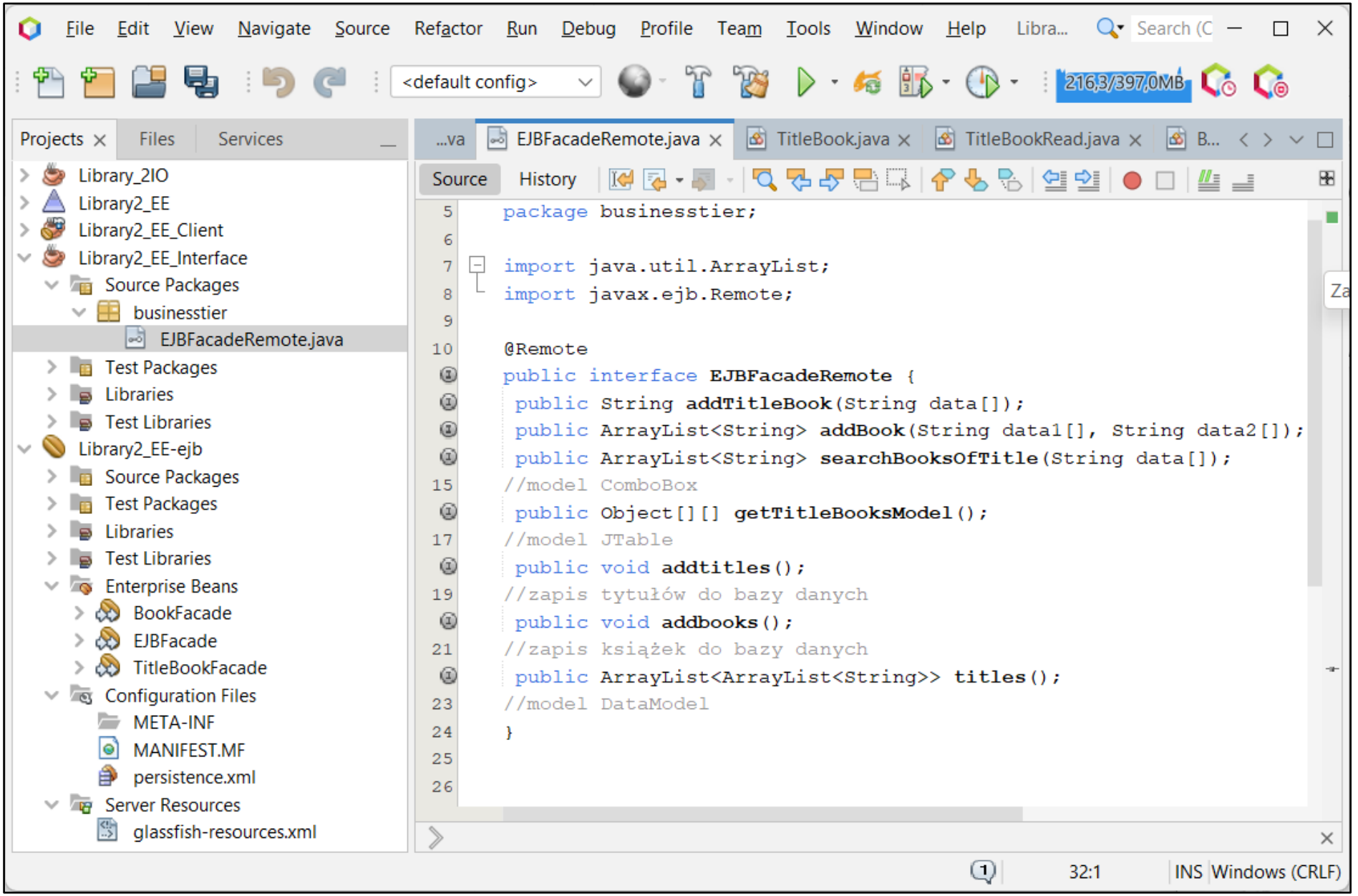
2. Klasa TitleBookFacade implementująca klasę AbstractFacade - kontroler typu Session Bean do utrwalania obiektów typu TitleBook i TitleBookRead



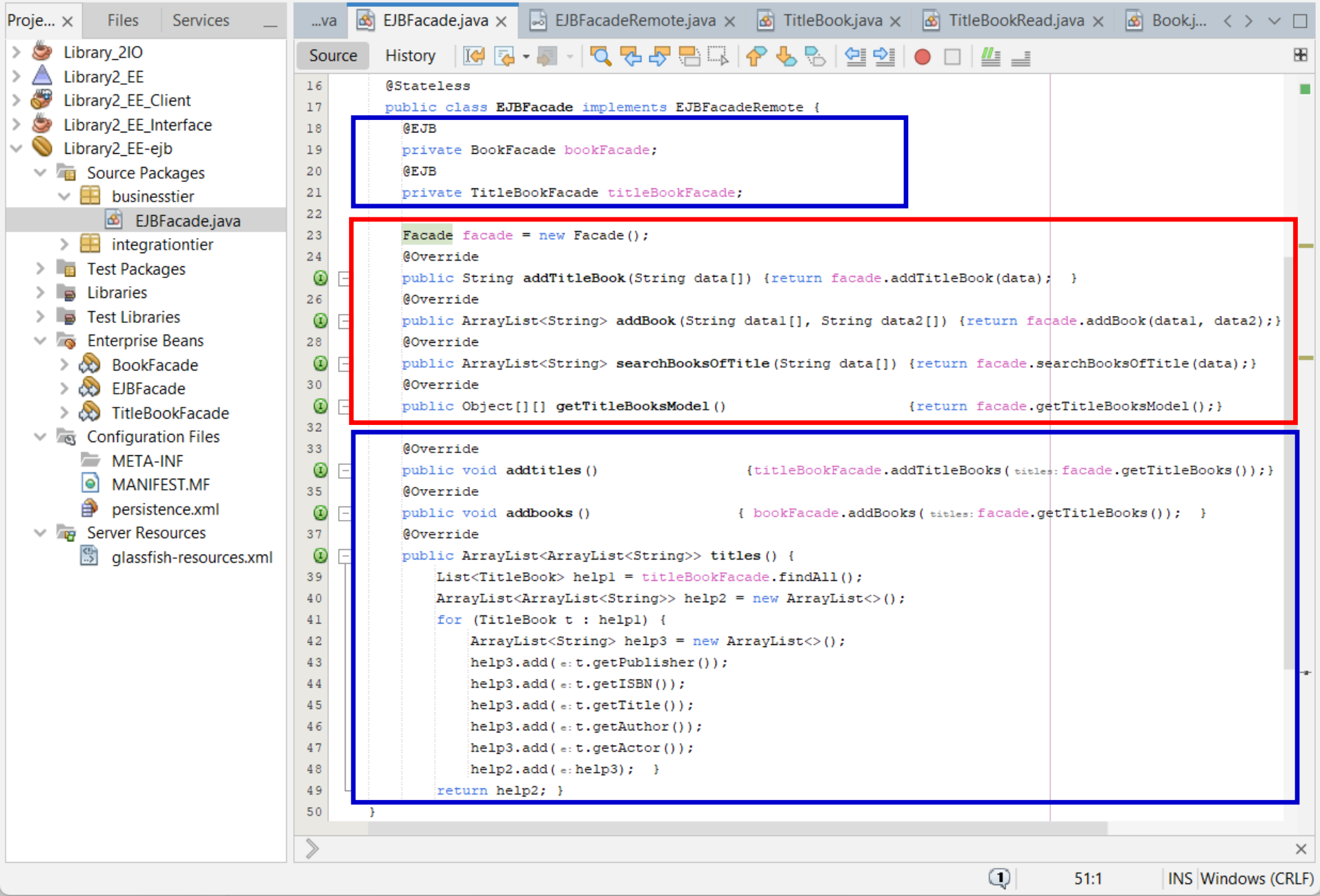
3. Klasa BookFacade implementująca klasę AbstractFacade - kontroler typu Session Bean do utrwalania obiektów typu Book



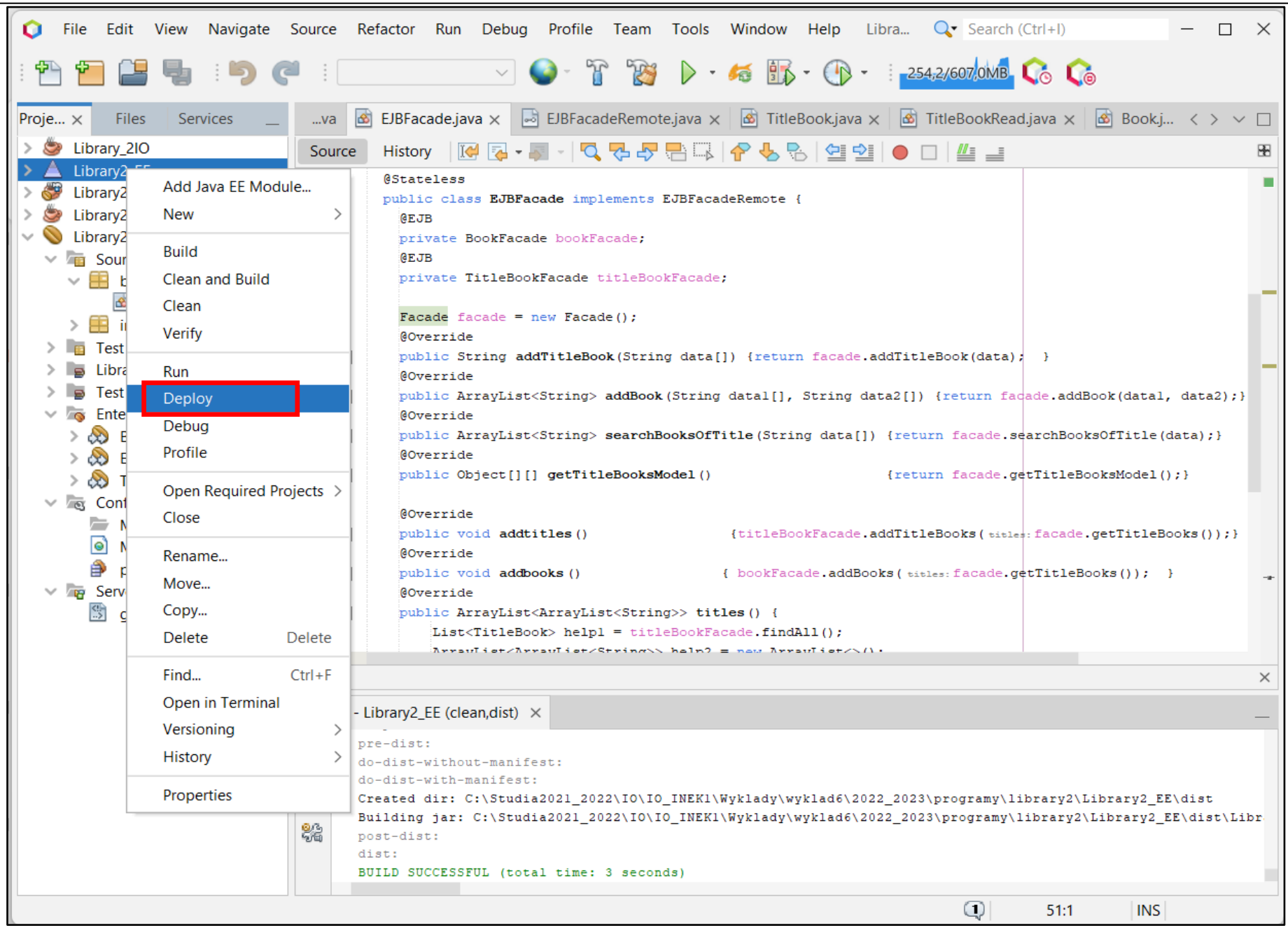
5. Uzupełnienie deklaracji metod o zdalnym dostępie do metod komponentów do utrwalania obiektów typu Entity w interfejsie komponentu typu Session Bean



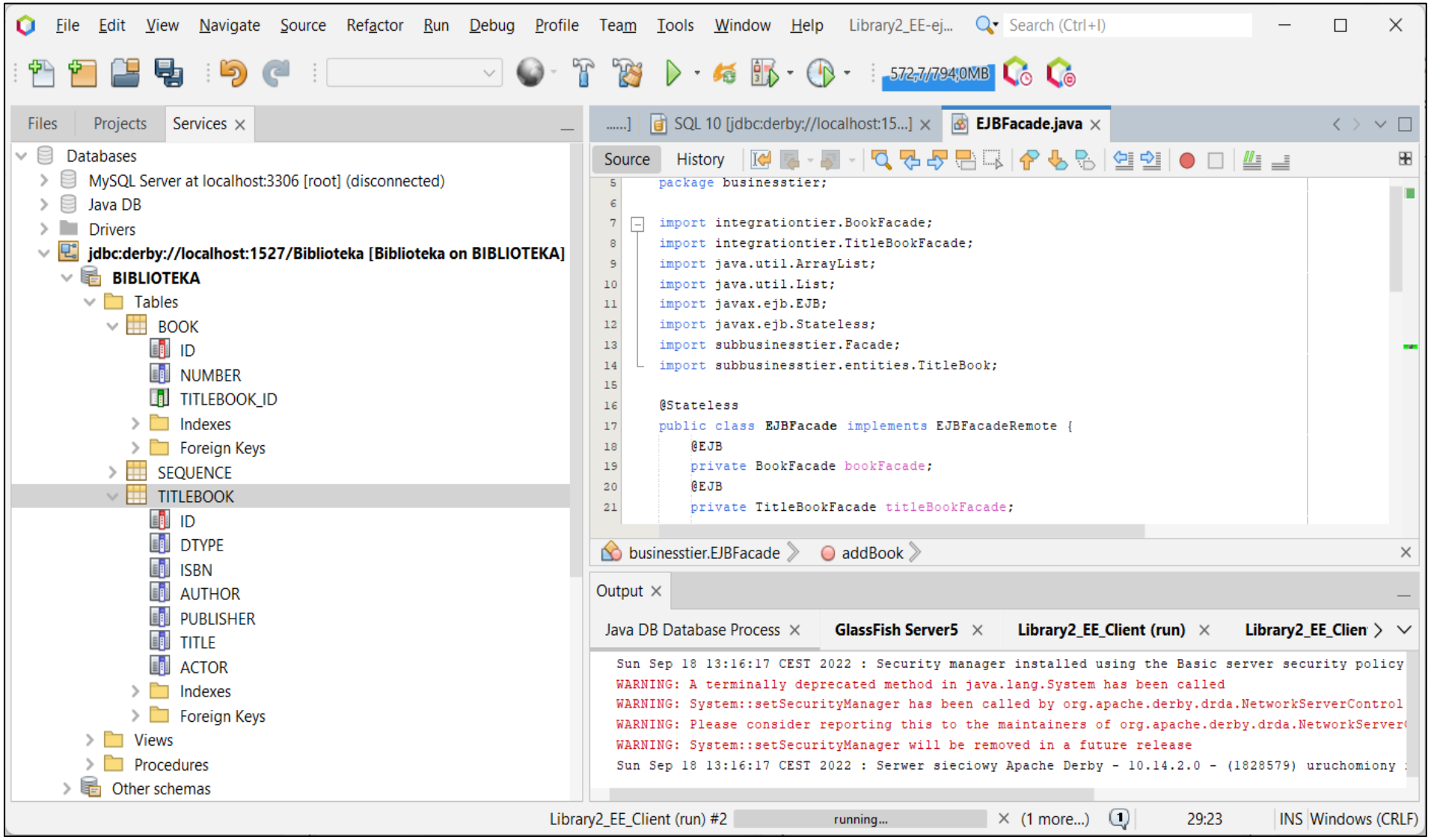
4. Klasa EJBFacade - główny komponent *Warstwy biznesowej* udostępnia metody logiki biznesowej i zarządza komponentami do utrwalania obiektów typu Entity.



5. Uruchomienie desktopowej aplikacji wielowarstwowej typu EE



6. Wygenerowanie pustych tabel w bazie danych



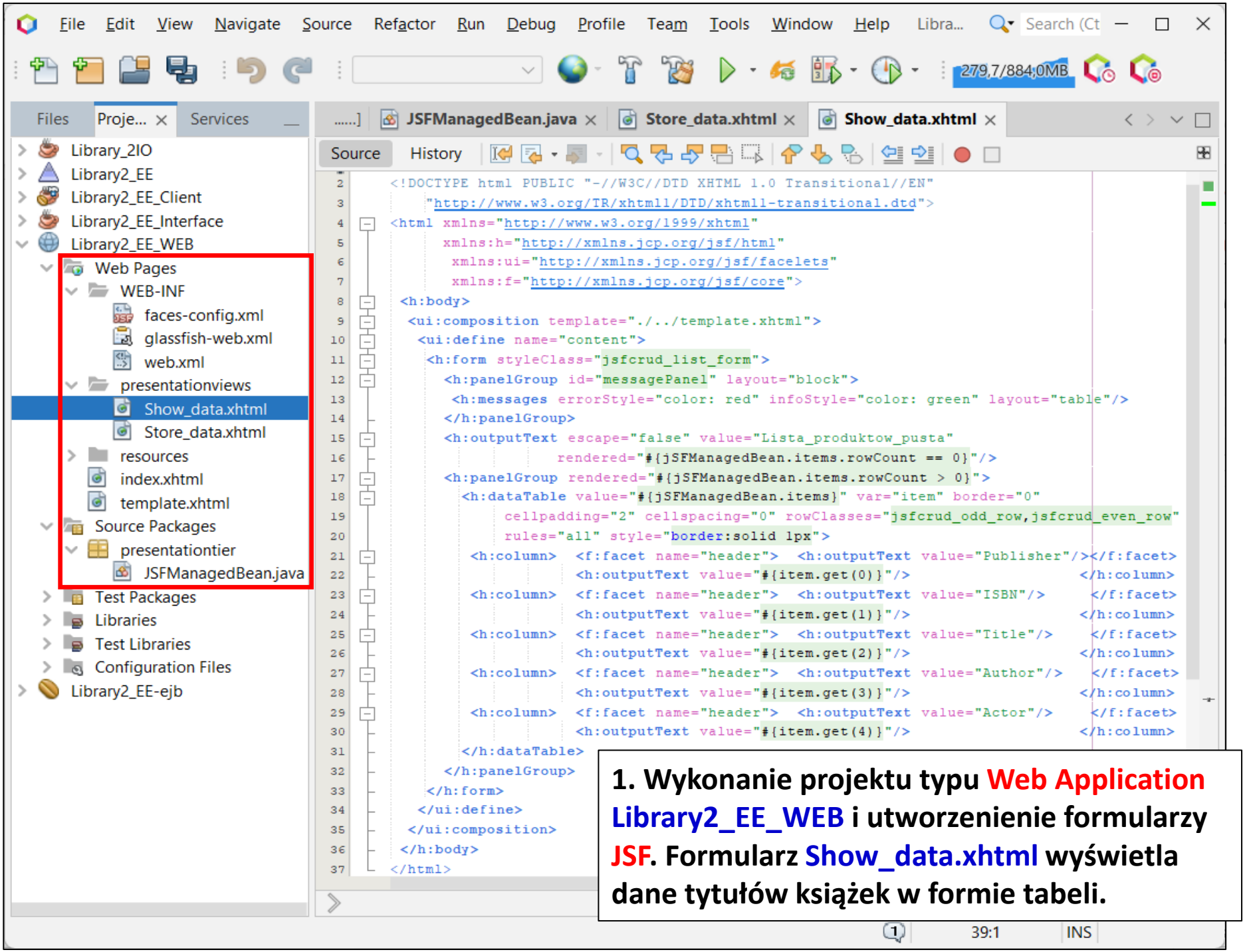
Zagadnienia

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

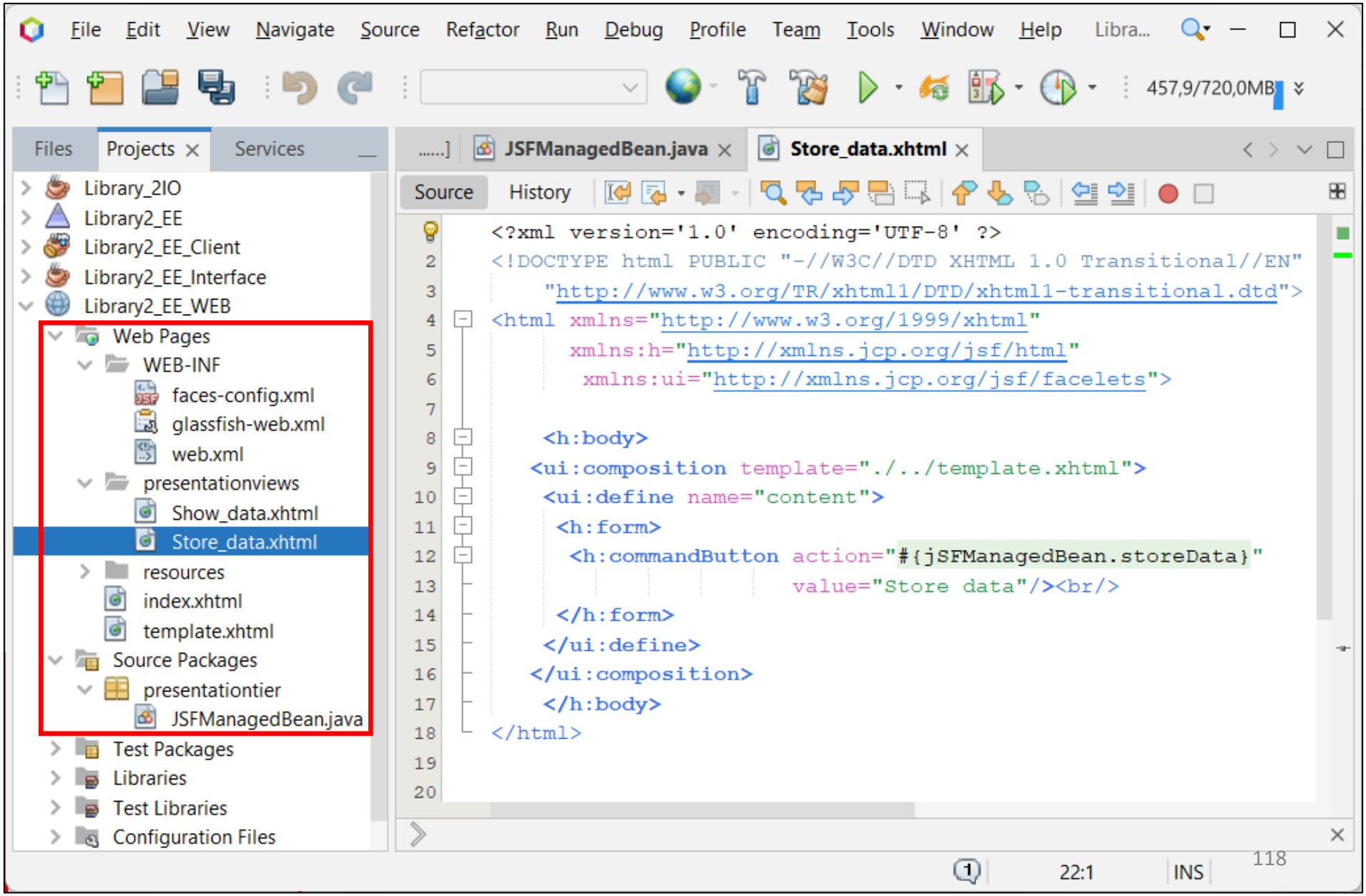
Architektura oprogramowania

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(businesstier)	Logika biznesowa, transakcje, dane i usługi
Warstwa integracji Library2_EE-ejb (integrationtier)	Adaptery zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania
Warstwa zasobów Baza danych Biblioteka typu Apache Derby	Zasoby, dane i usługi zewnętrzne

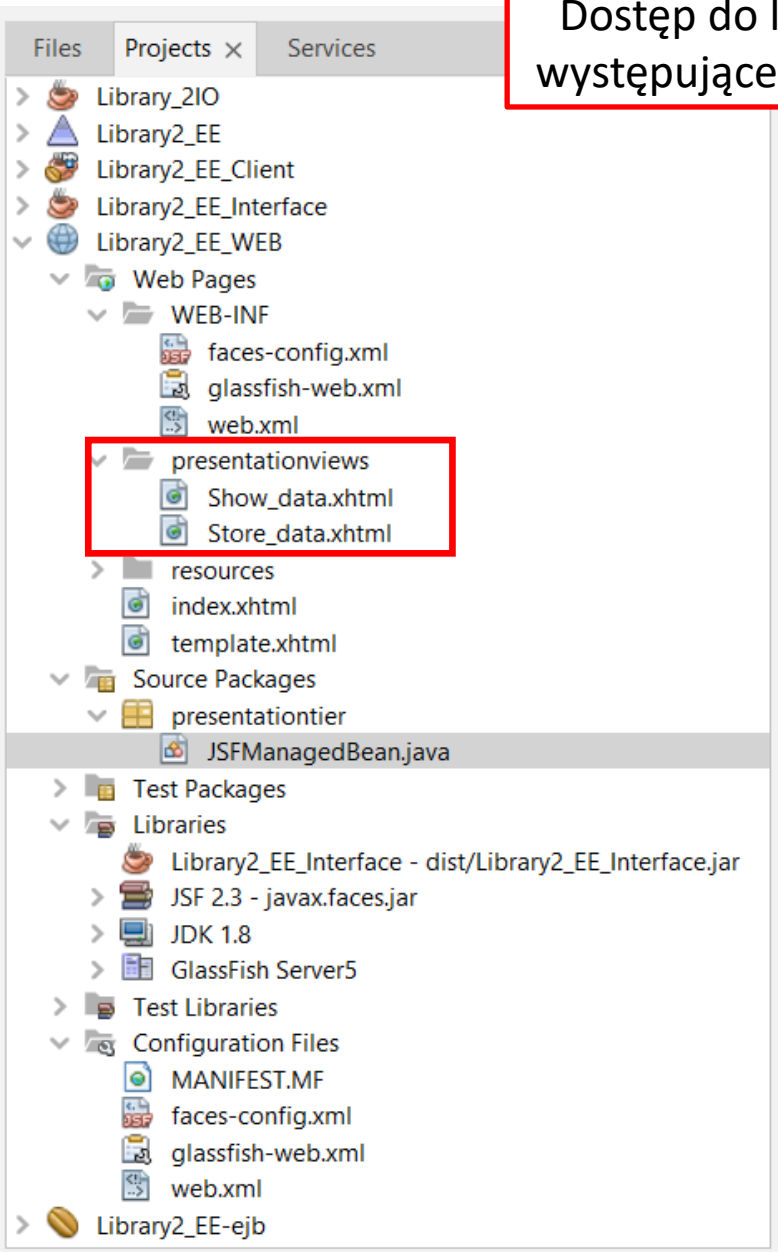

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
279,7/884,0MB
Files Proje... Services
Library_2IO
Library2_EE
Library2_EE_Client
Library2_EE_Interface
Library2_EE_WEB
Web Pages
WEB-INF
faces-config.xml
glassfish-web.xml
web.xml
presentationviews
Show_data.xhtml
Store_data.xhtml
resources
index.xhtml
template.xhtml
Source Packages
presentationtier
JSFManagedBean.java
Test Packages
Libraries
Test Libraries
Configuration Files
Library2_EE-ejb
JSFManagedBean.java
Store_data.xhtml
Show_data.xhtml
Source History
<!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 utworzenienie formularzy JSF. Formularz Show_data.xhtml wyświetla dane tytułów książek w formie tabeli.
39:1
INS

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



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

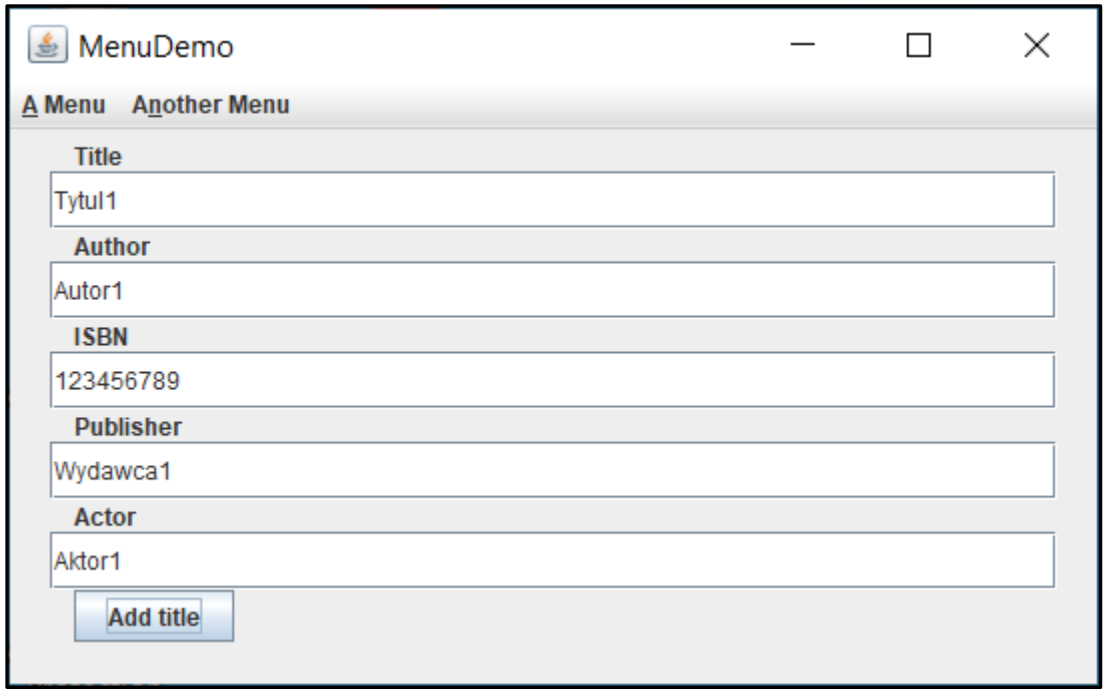
Dostęp do logiki biznesowej za pomocą wzorca **SessionFacade**, występującego również w roli wzorca **Singleton (komponent EJB)**

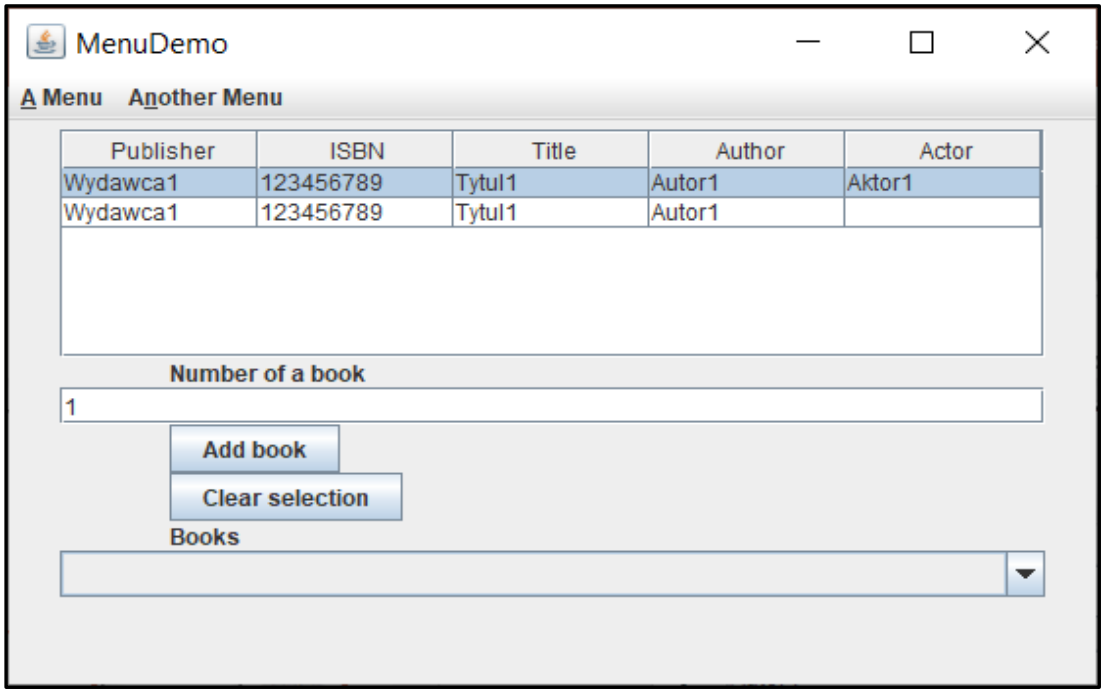



```
5  package presentationtier;
6
7  import businesstier.EJBFacadeRemote;
8  import javax.inject.Named;
9  import javax.ejb.EJB;
10 import javax.enterprise.context.RequestScoped;
11 import javax.faces.model.DataModel;
12 import javax.faces.model.ListDataModel;
13
14 @Named(value = "jSFManagedBean")
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     }
```

presentationtier.JSFManagedBean › getItems

40:27 INS Windows (CRLF)

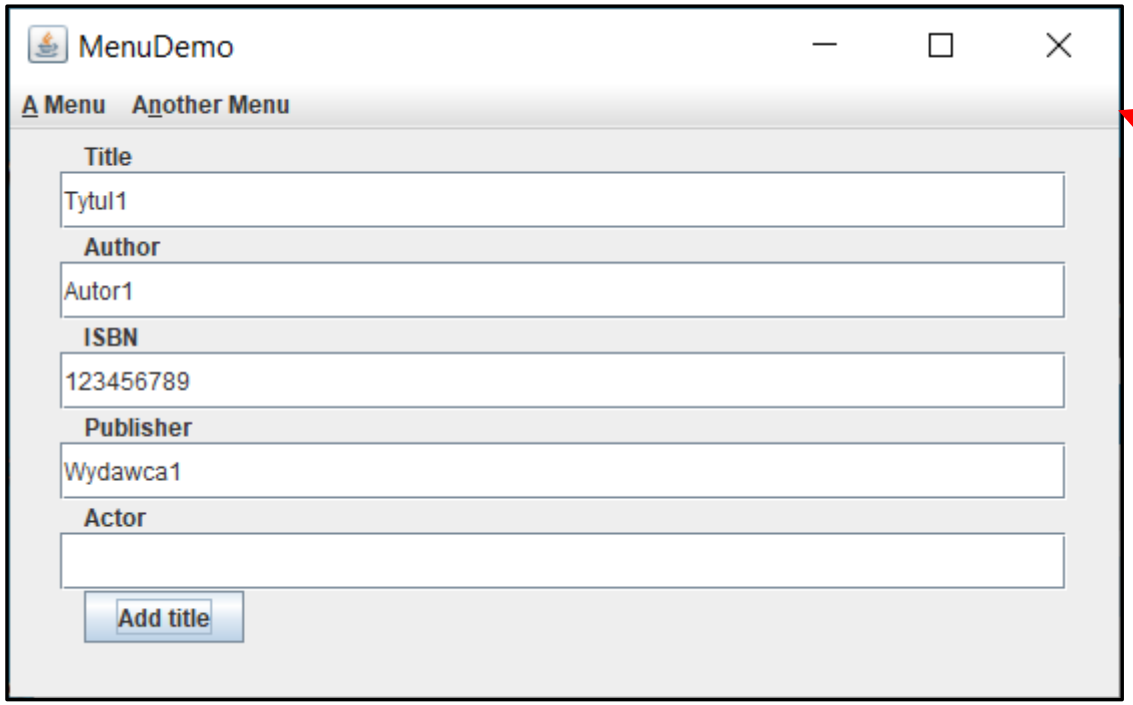


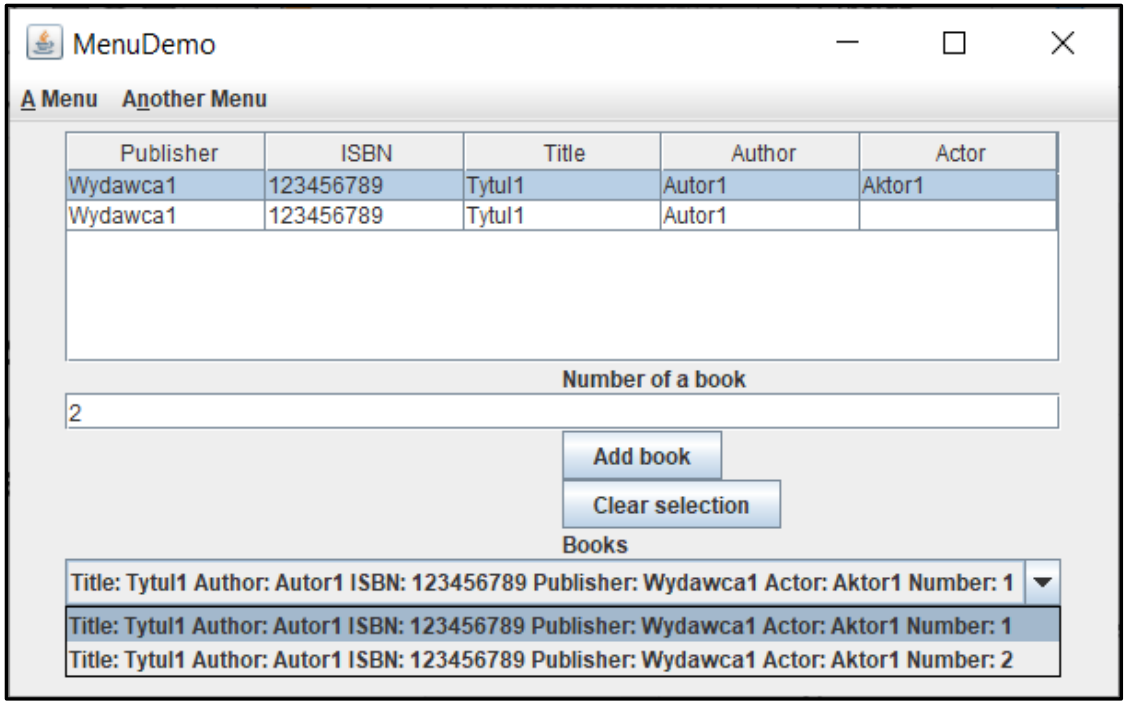


4. Uruchomienie bazodanowej aplikacji typu Enterprise Library2_EE na platformie JavaEE za pomocą operacji Deploy zawierającej dwa typy aplikacji klienckiej: desktopową Library2_EE_Client i internetową Library2_EE_WEB.

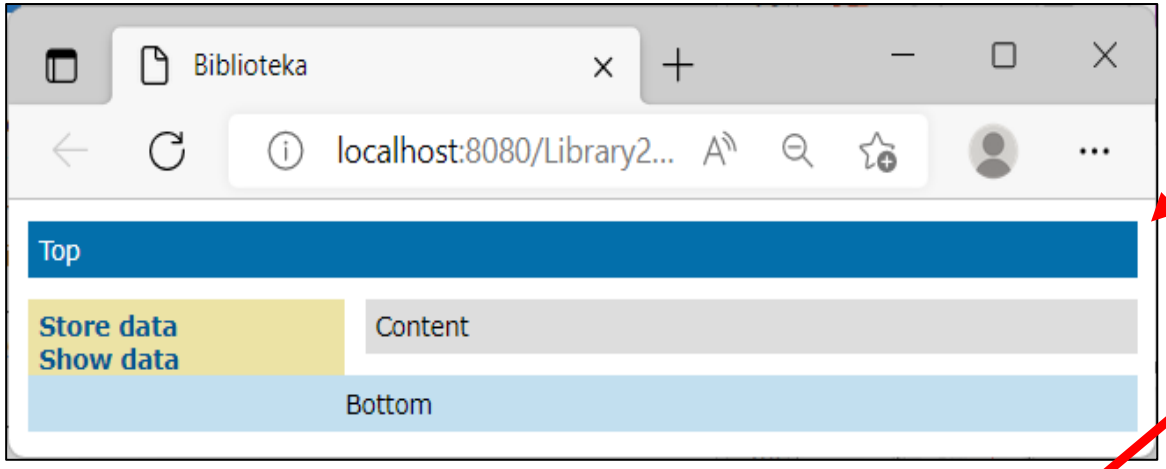
Uruchomiono 2 instancje aplikacji desktopowej za pomocą operacji Run oraz jedną instancję aplikacji internetowej również za pomocą operacji Run.

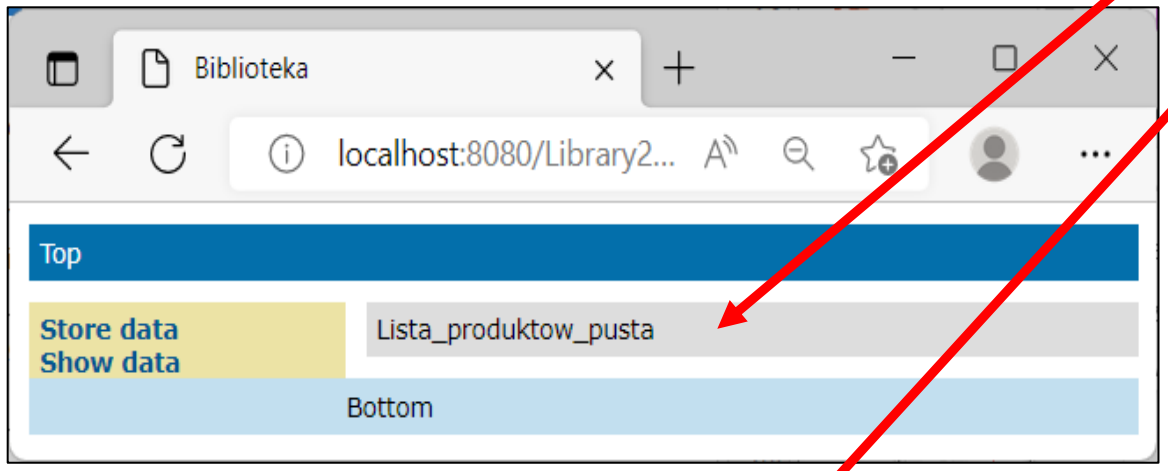
1-aplikacja desktopowa

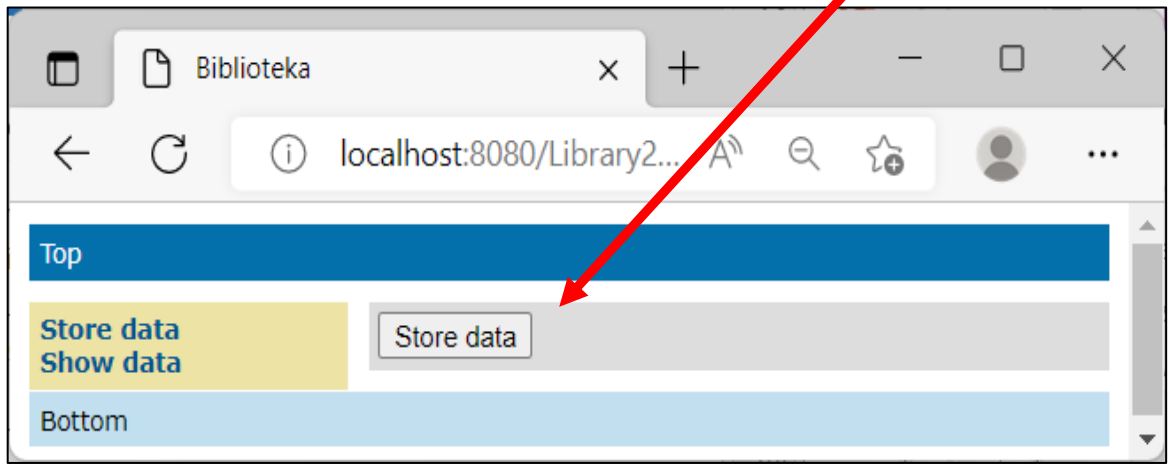




4 (cd).
Uruchomiona 2-a aplikacja desktopowa



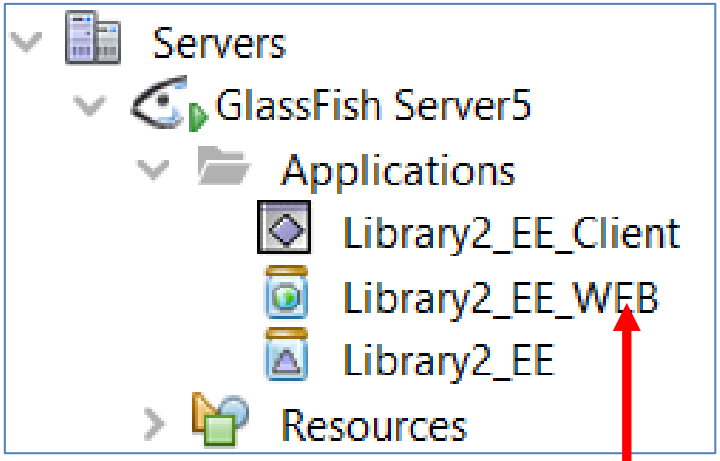




5. Uruchomiona 1-a aplikacja internetowa za pomocą operacji Run.

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

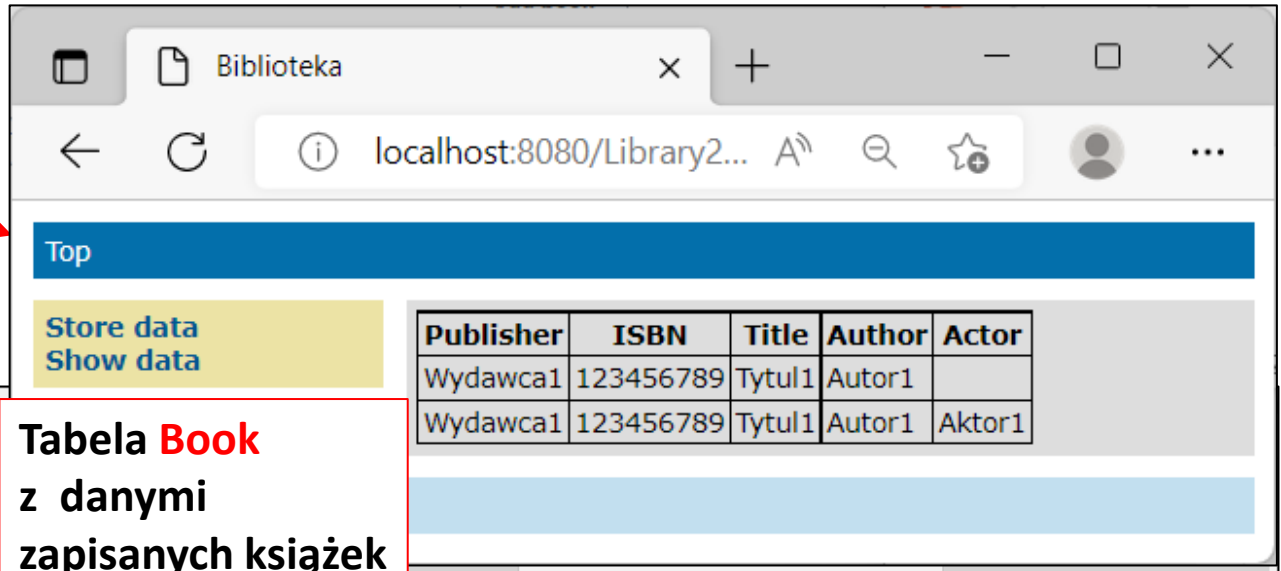
Formularz Store data służy do zapisania danych tytułów i książek przechowywanych w *Warstwie biznesowej* (projekt Library_2IO) do bazy danych

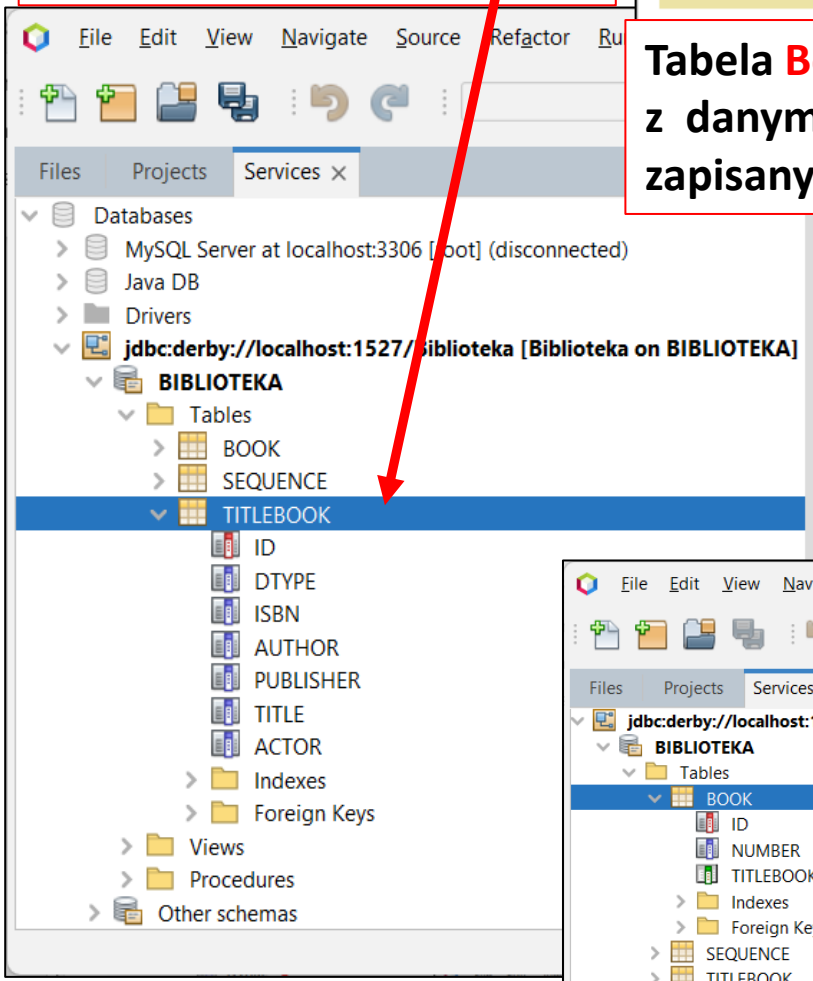


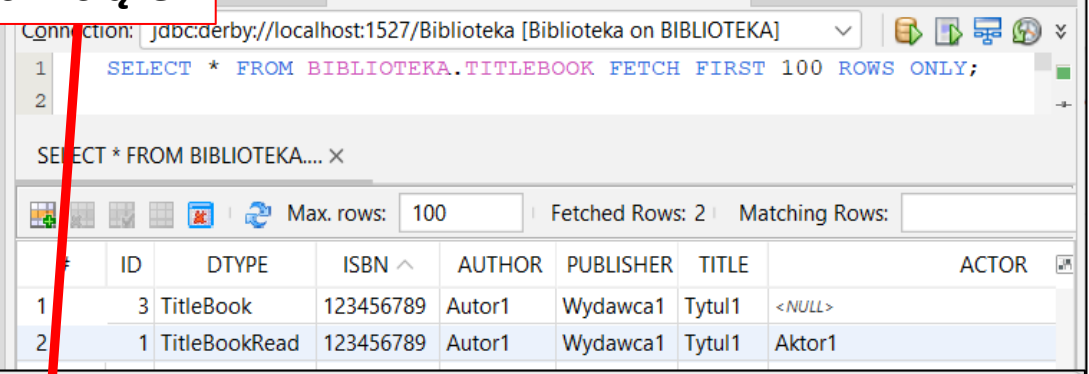
Po uruchomieniu 1-ej aplikacji internetowej Library2_EJB_WEB na serwerze utworzył się kod tej aplikacji używany dla kolejno uruchomionych aplikacji internetowych

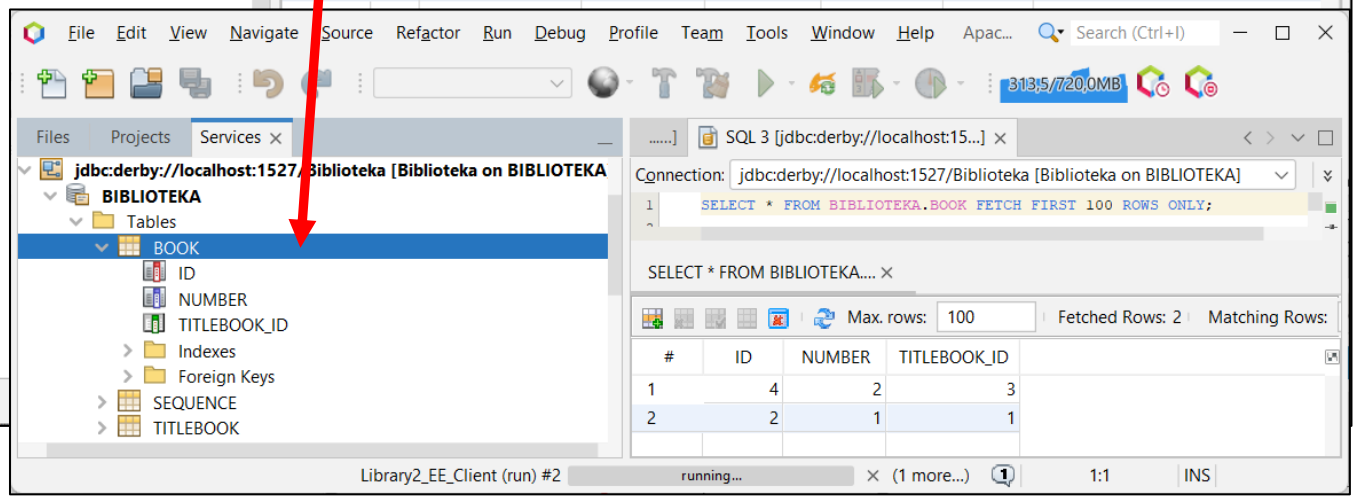
6. Uruchomiana aplikacja internetowa – formularz Show data służy do wyświetlenia danych tytułów pobranych z tabeli TitleBook.

Tabela Book z danymi zapisanych książek









Zagadnienia

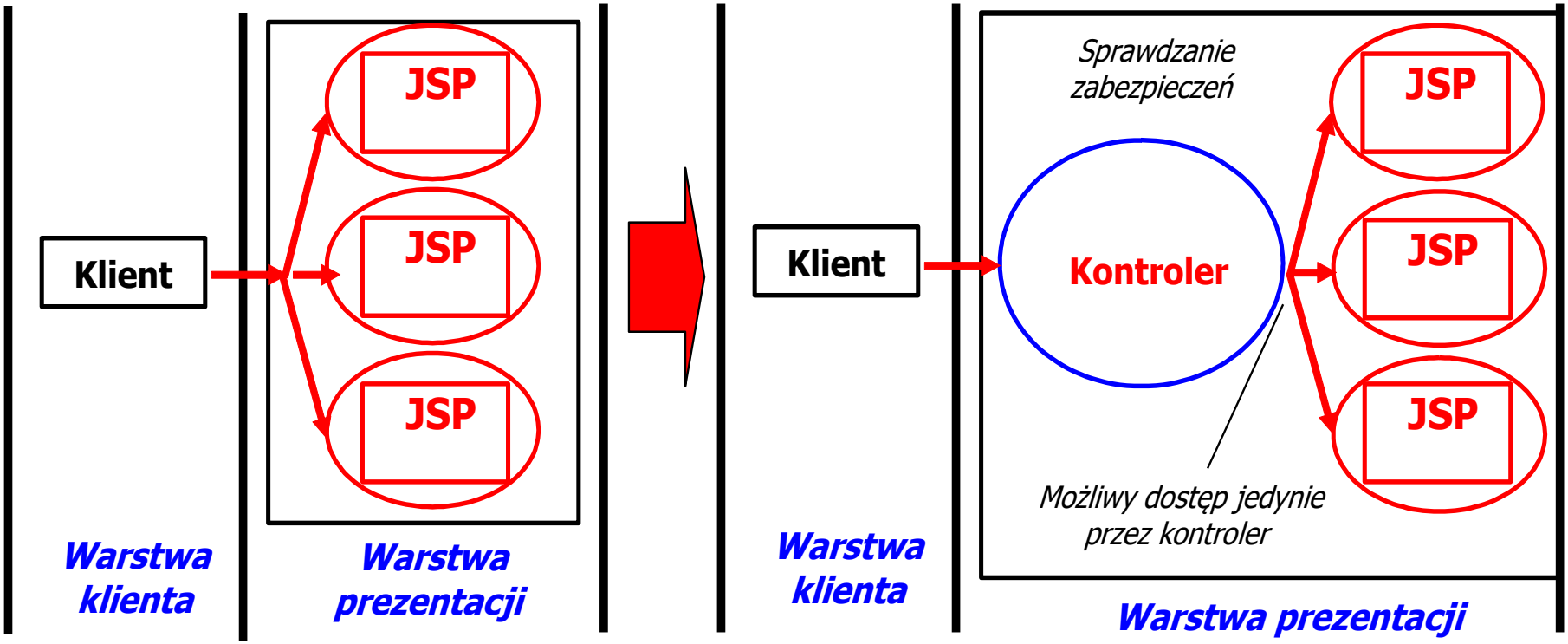
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. 5 zasad programowania solidnego (solid) [5]
5. Przykład modelowania i projektowania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

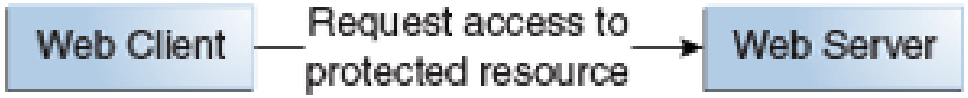
1. Dodanie zabezpieczeń typu Security

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

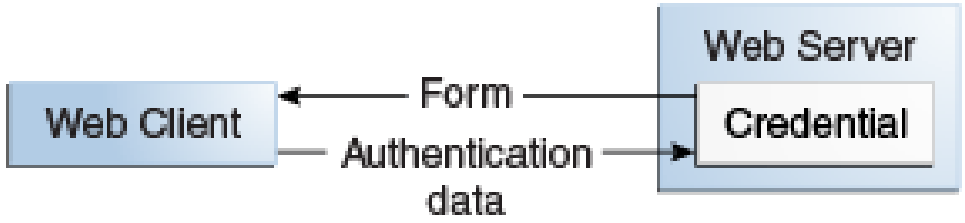


Przebieg uwierzytelniania i autoryzacji

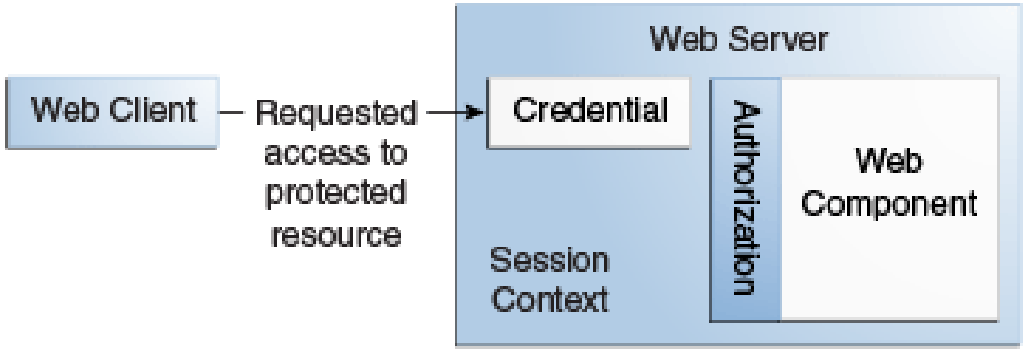
https://javaee.github.io/tutorial/security-intro001.html



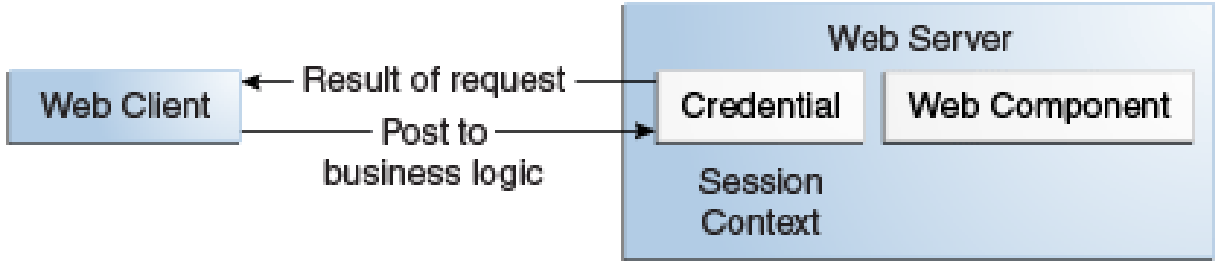
1. Początkowe żądanie do dostępu do chronionych zasobów



2. Pobranie danych do **uwierzytelnienia**



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

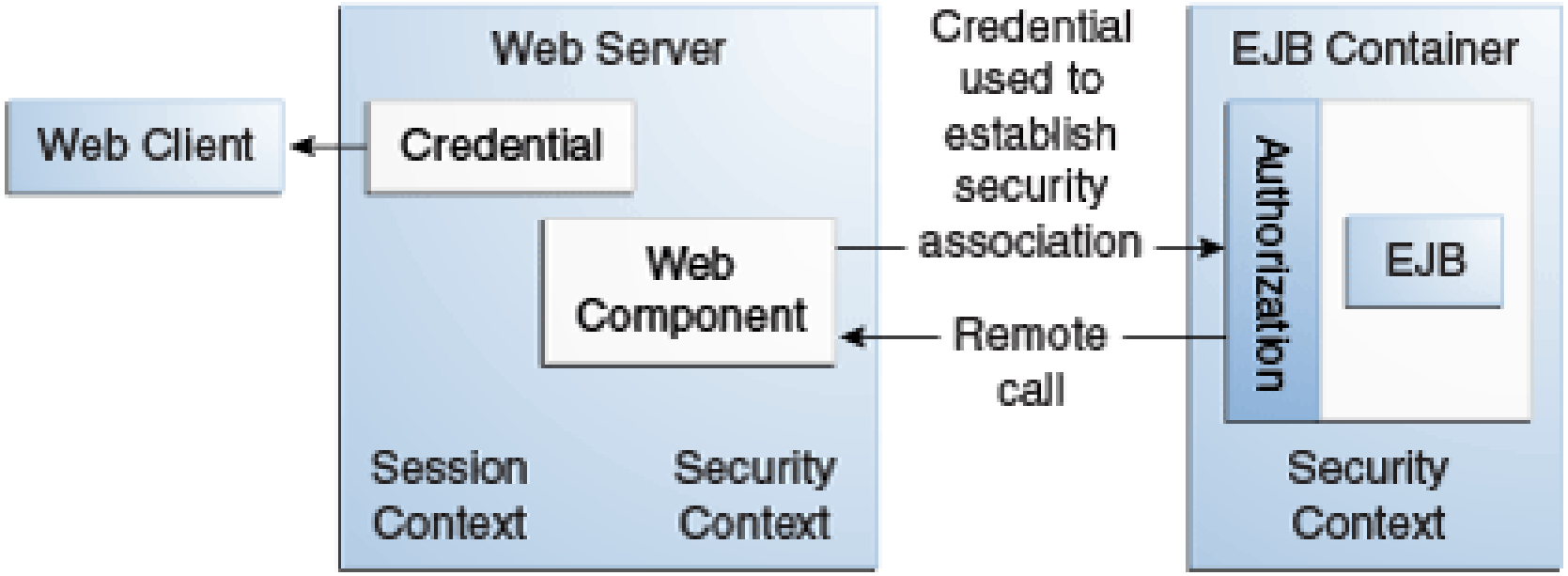


4. Przesłanie wyniku kontroli dostępu. Wynik pozytywny - wysłanie żądania do kontroli dostępu do ***Warstwy logiki biznesowej***

Przebieg uwierzytelniania i autoryzacji (cd)

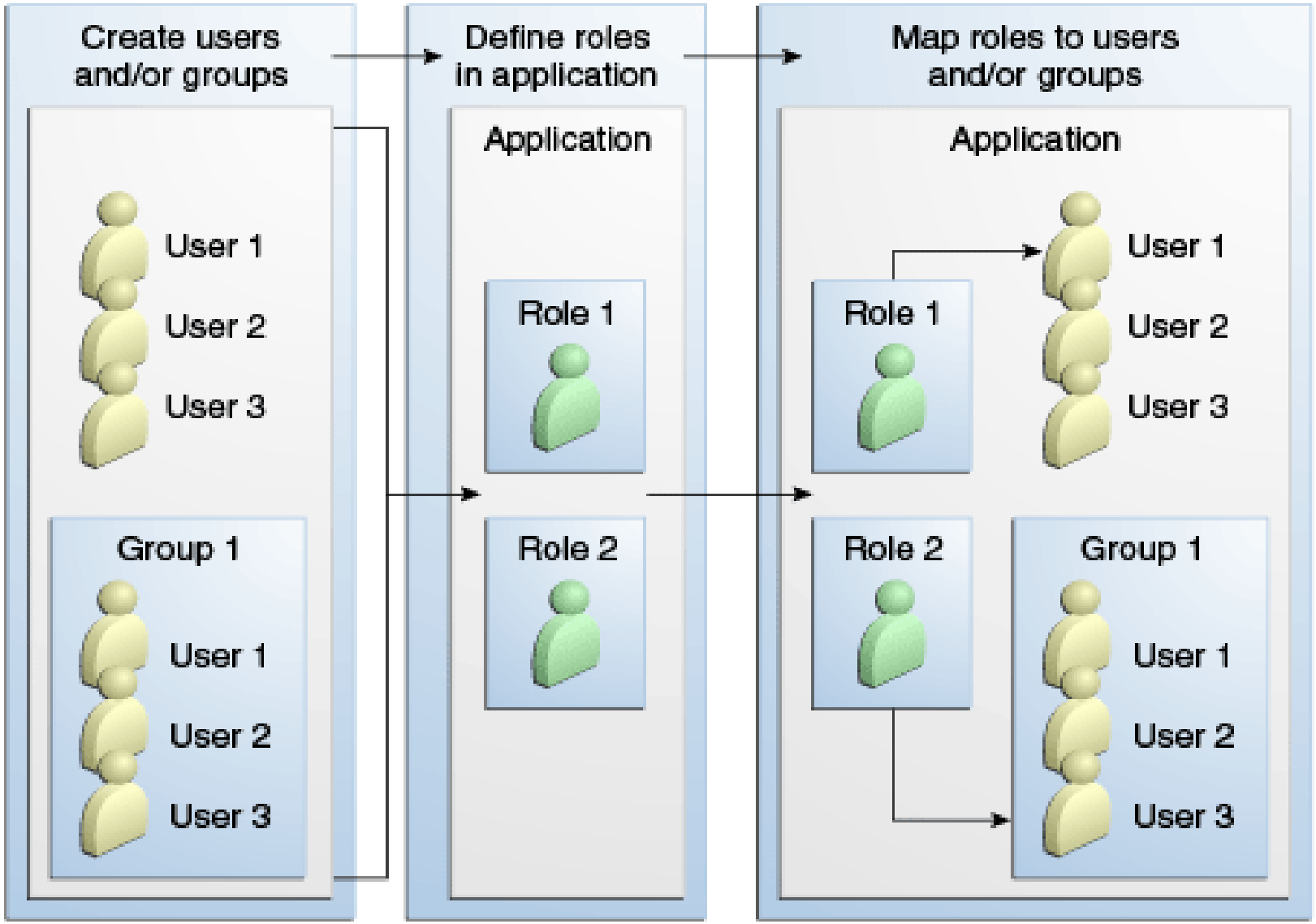
Przebieg autoryzacji w dostępie do *Warstwy logiki biznesowej*

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

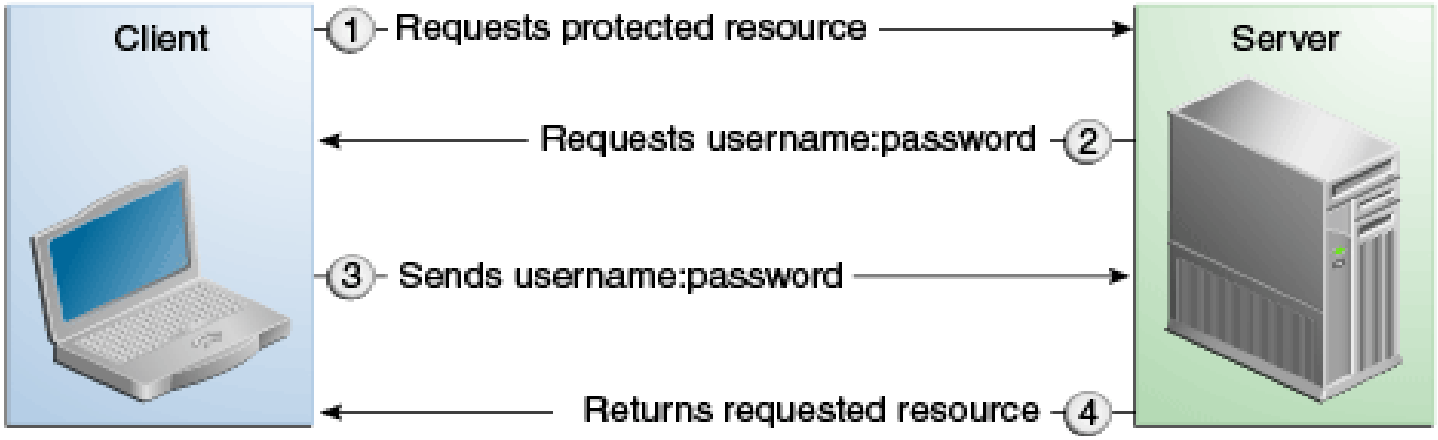


Bazy użytkowników i grup użytkowników. Role typu security przypisane do użytkowników i grup

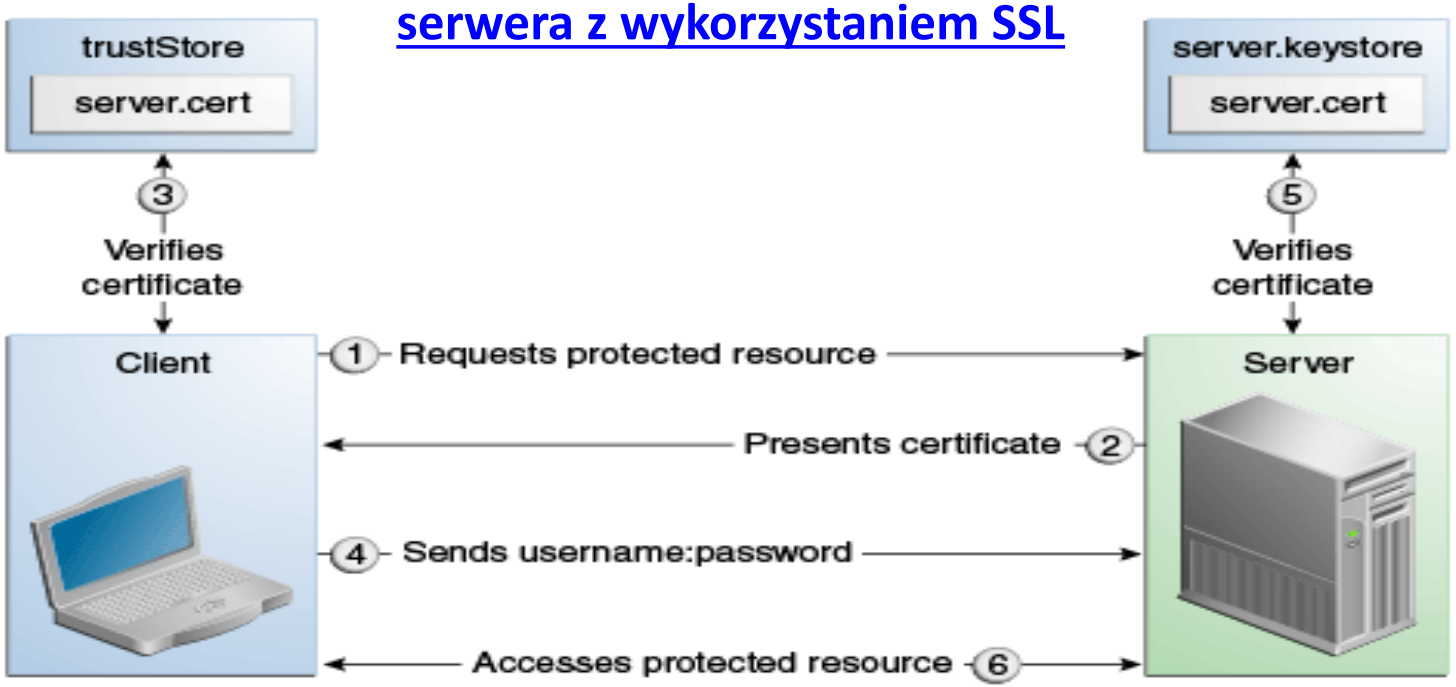
https://javaee.github.io/tutorial/security-intro005.html



Przebieg logowania oparty na nazwie użytkownika i haśle

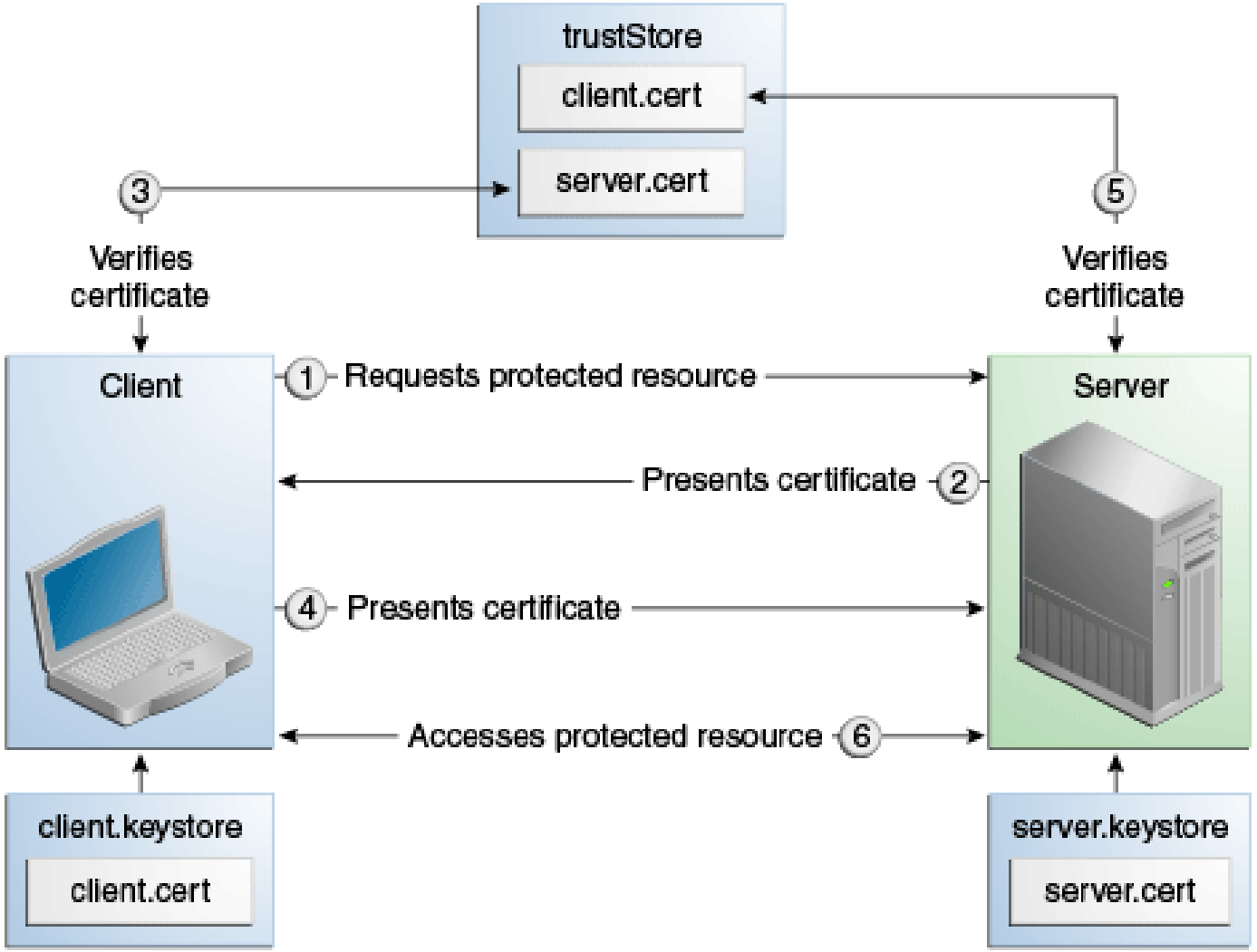


Przebieg logowania oparty na nazwie użytkownika i haśle oraz certyfikacie serwera z wykorzystaniem SSL



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 informację specyfikującą role bezpieczeństwa i wymagania dostępu są mapowane w role specyficzne dla środowiska oraz użytkowników i polisy 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")

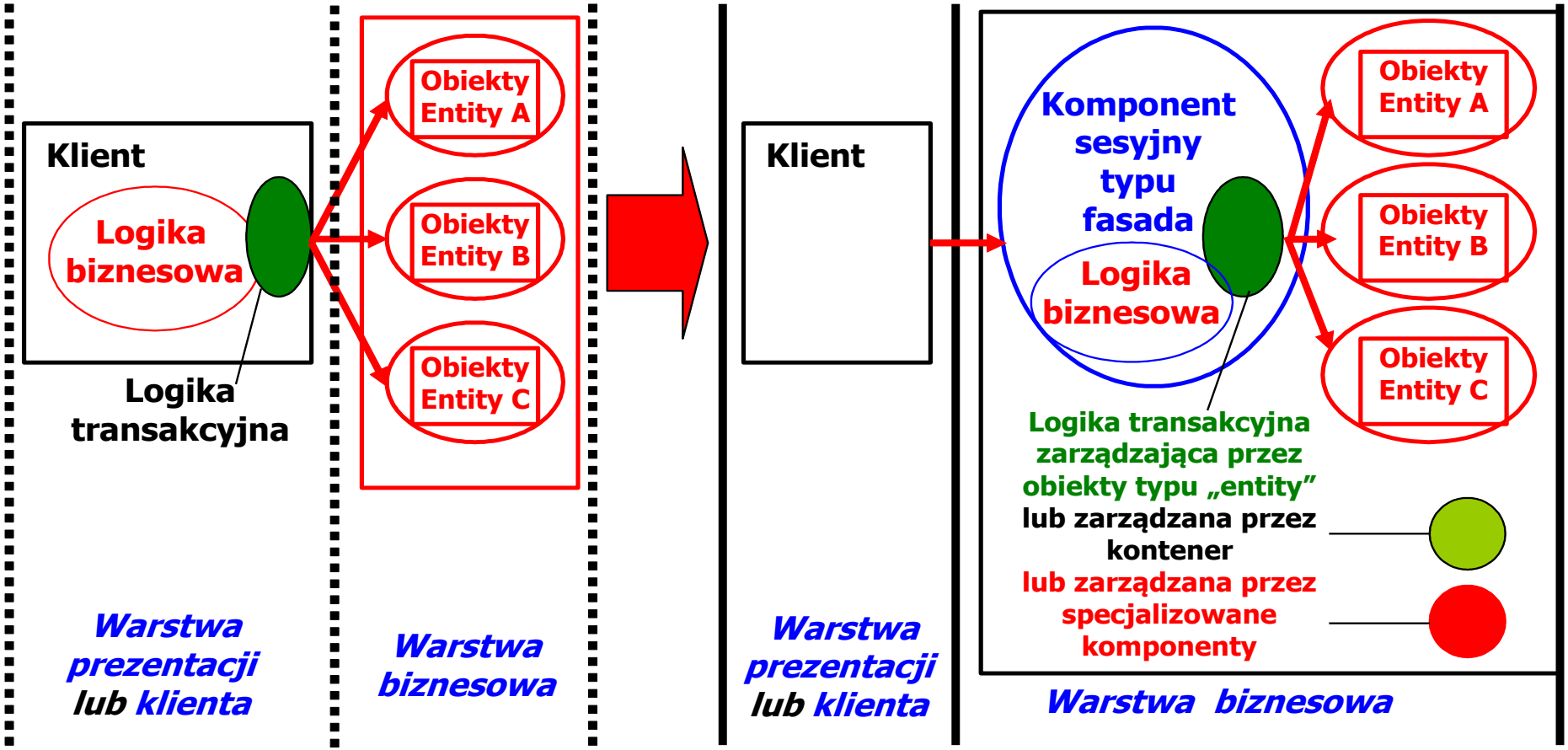
public class Page1 extends AbstractPageBean { //... }

- **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)

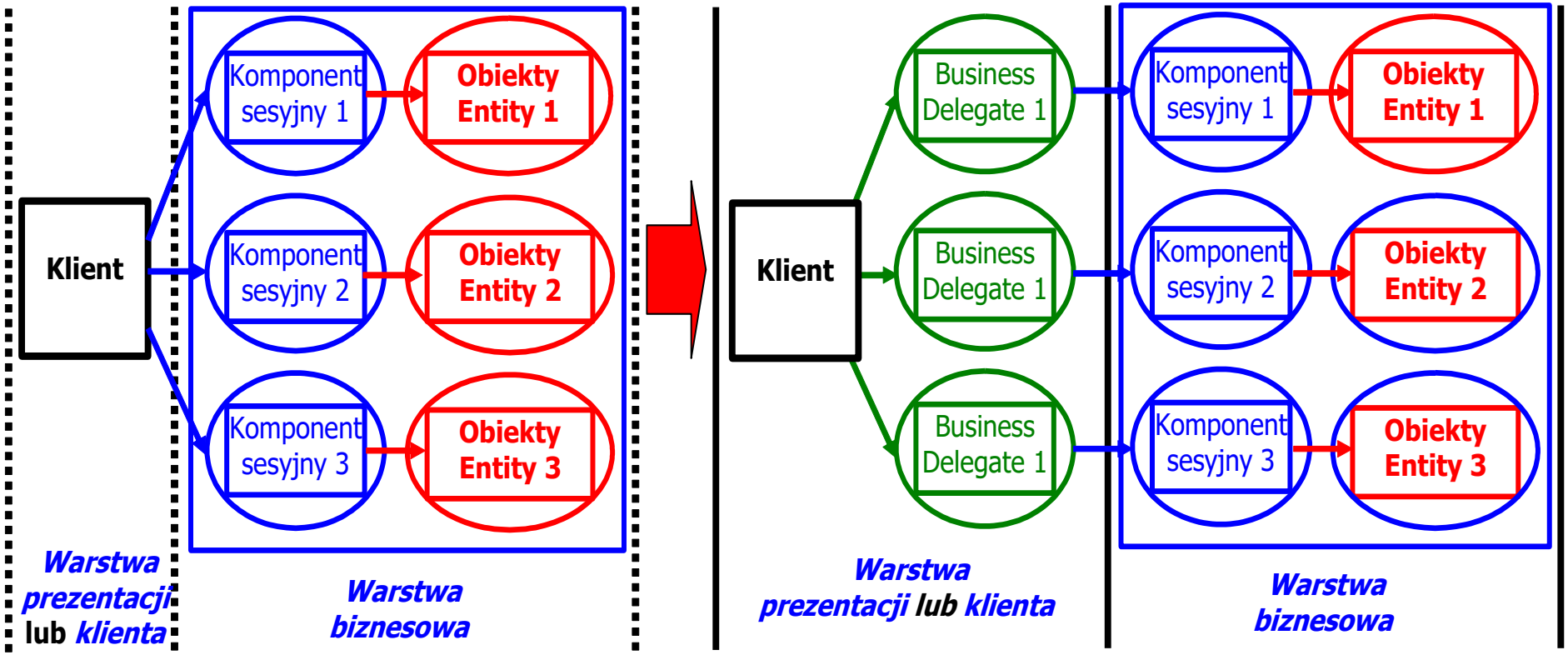
Refaktoryzacja warstwy biznesowej 1

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)**



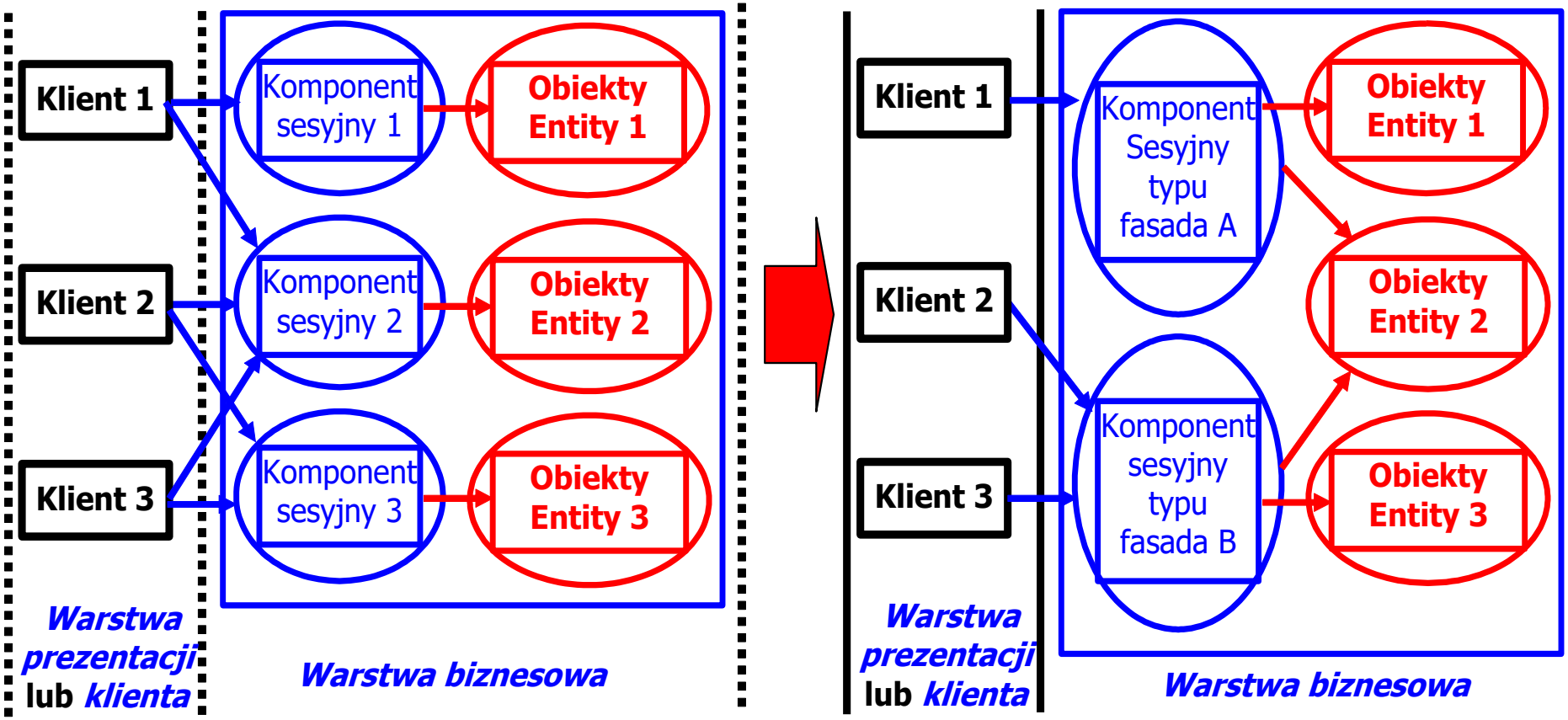
Refaktoryzacja *Warstwy biznesowej* 2

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

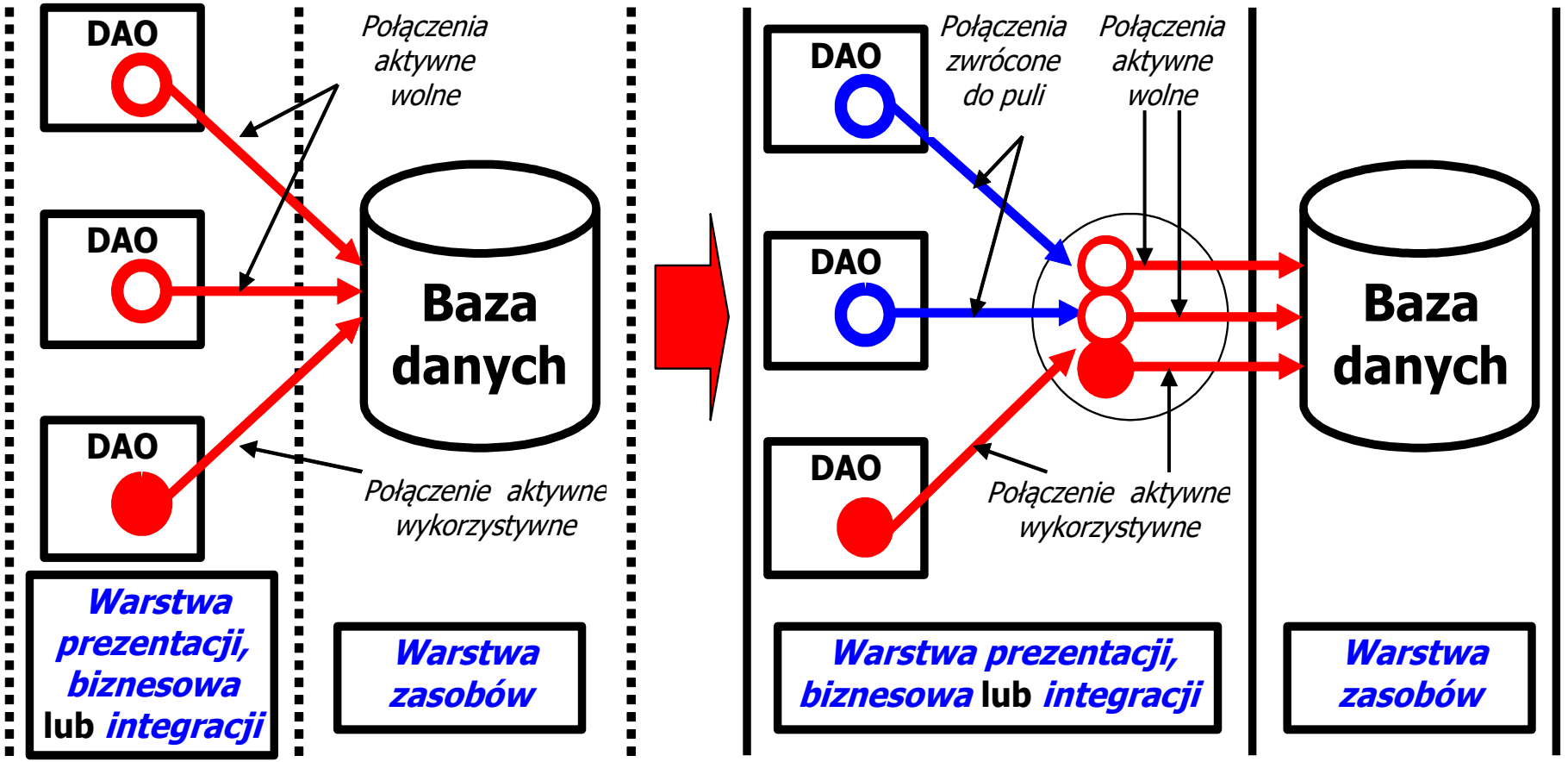
Sesyjne komponenty fasadowe typu Control (każdy komponent jako odrębna usługa biznesowa), hermetyzujące **obiekty danych typu Entity** z ***Warstwy biznesowej*** są udostępniane klientom w innych warstwach. Zwykłe obiekty sesyjne są jedynie pośrednikami obiektów **Entity,** natomiast hermetyzują całe 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**