

**Wybrane aspekty projektowania
- budowa wielowarstwowego
modelu implementacji,
zastosowanie wzorców
projektowych
Wykład 7– część 2**

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, 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 2008
6. D. Alur, J. Crupi, D. Malks, Core J2EE. Wzorce projektowe
7. <https://docs.oracle.com/javaee/7/JEETT.pdf>

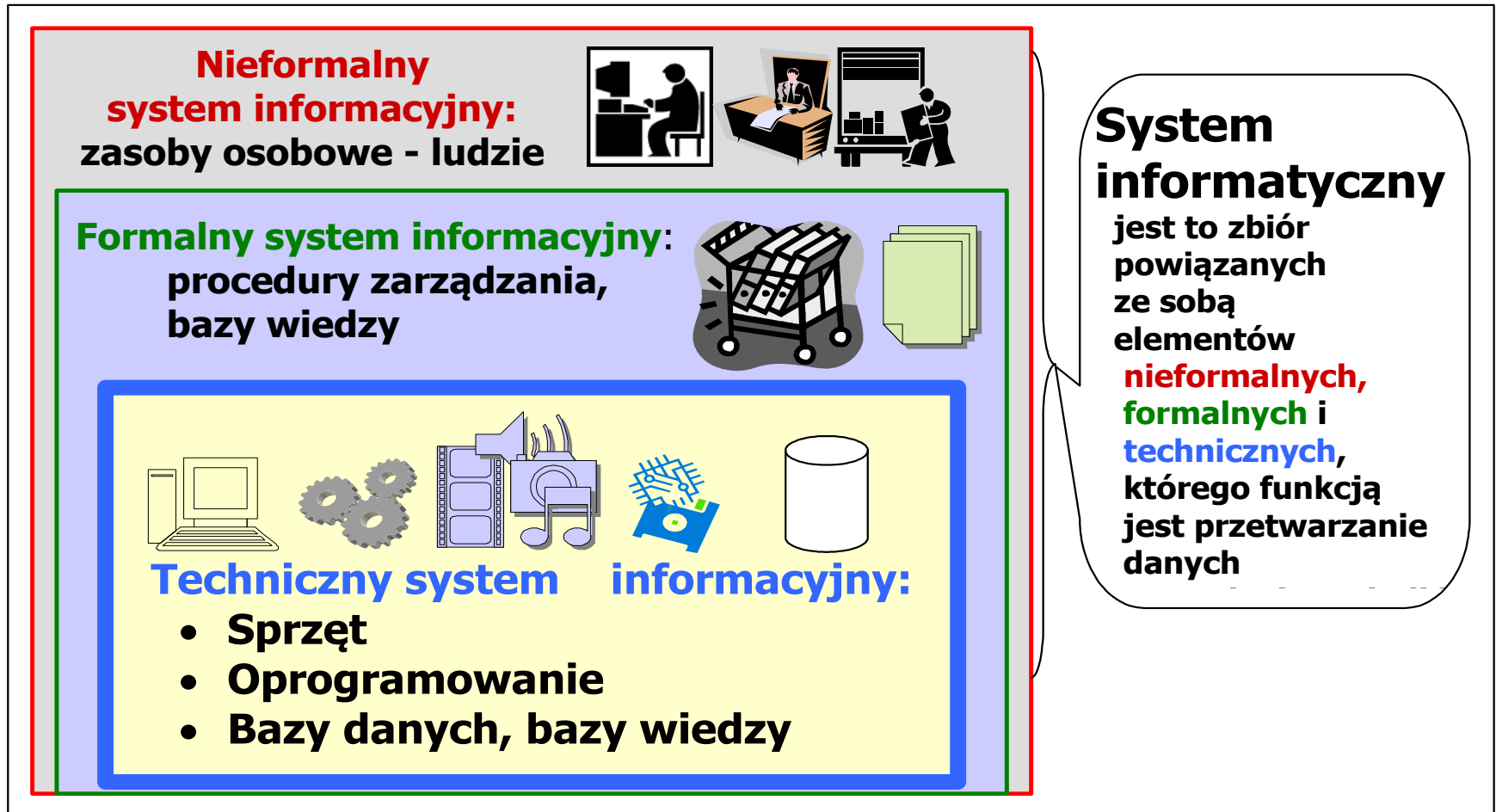
Zagadnienia

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

Zagadnienia

1. Wielowarstwowa architektura systemu informatycznego

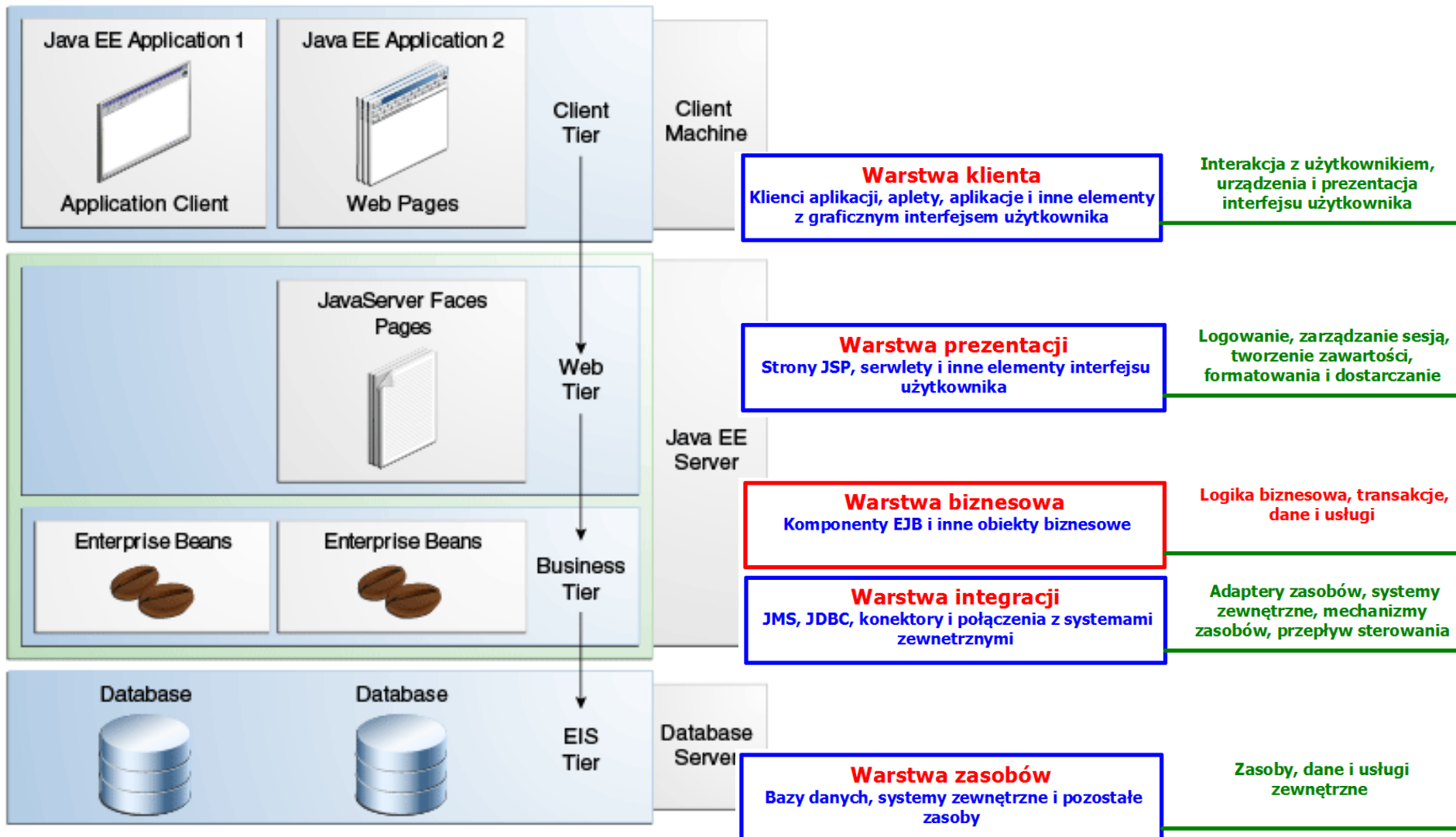
Definicja systemu informatycznego (wykład 1)



Techniczny system informacyjny

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

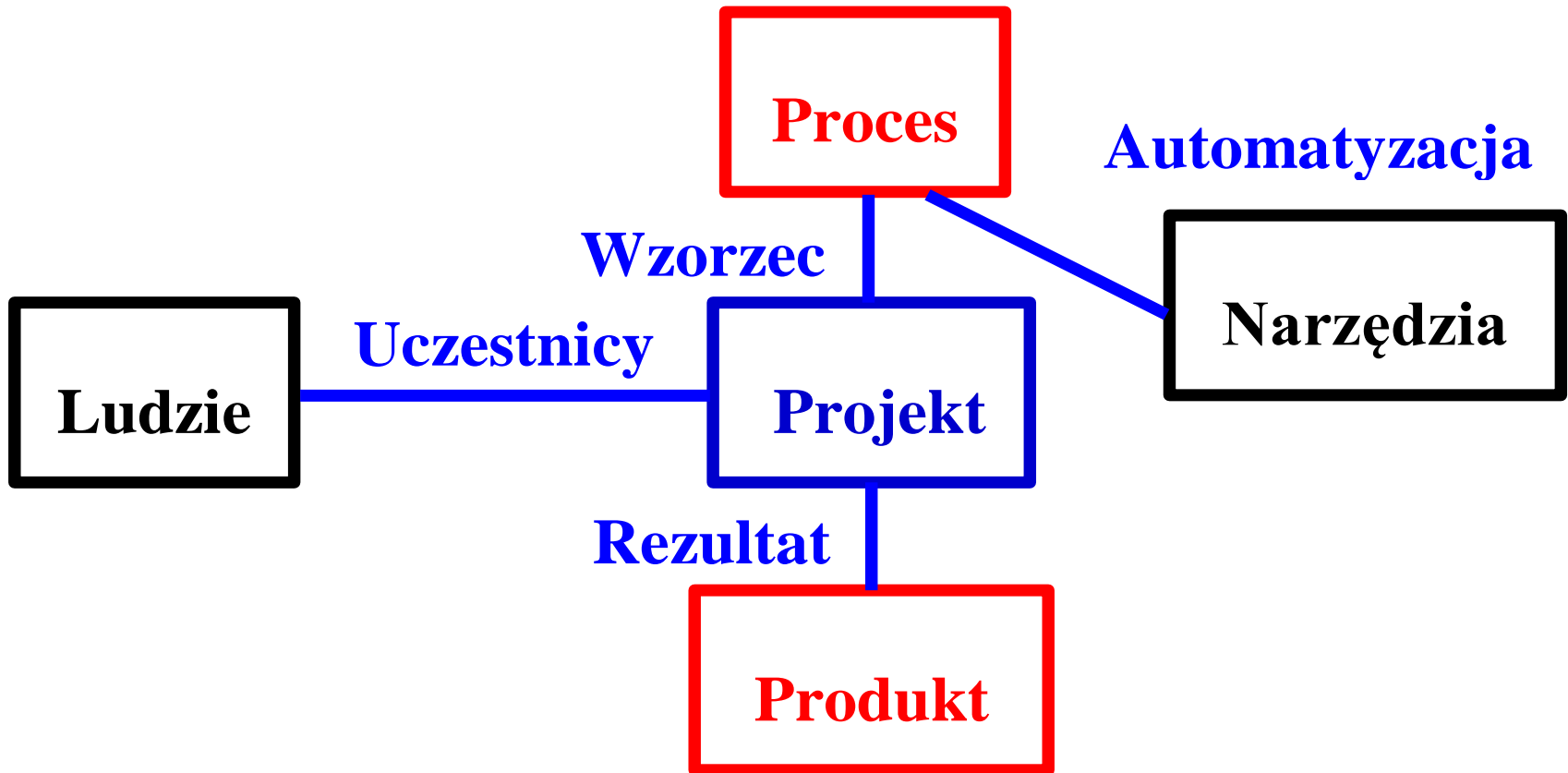
Java EE 7: <https://docs.oracle.com/javasee/7/tutorial> Pięciowarstwowy model logicznego rozdzielania zadań [6]



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 [6]**

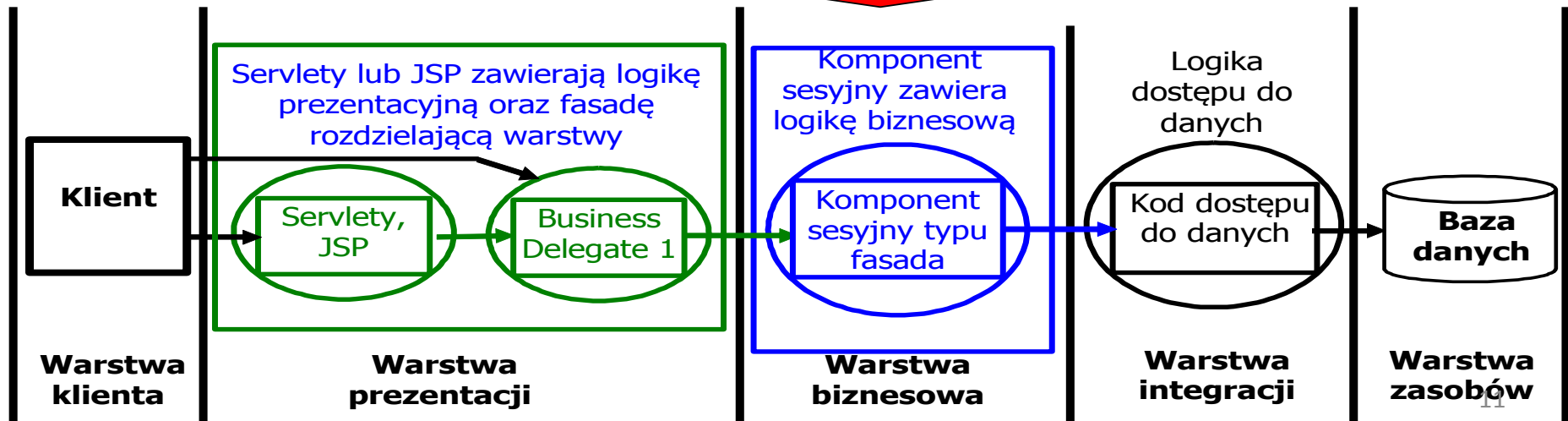
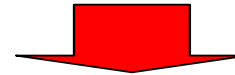
Ocena oprogramowania – metryki [2]

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**

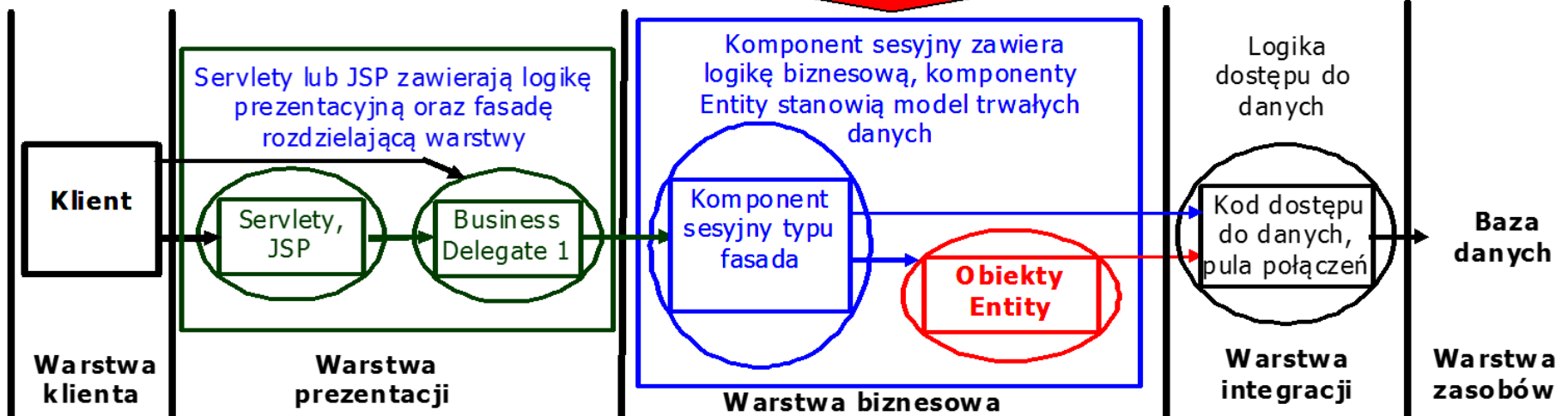
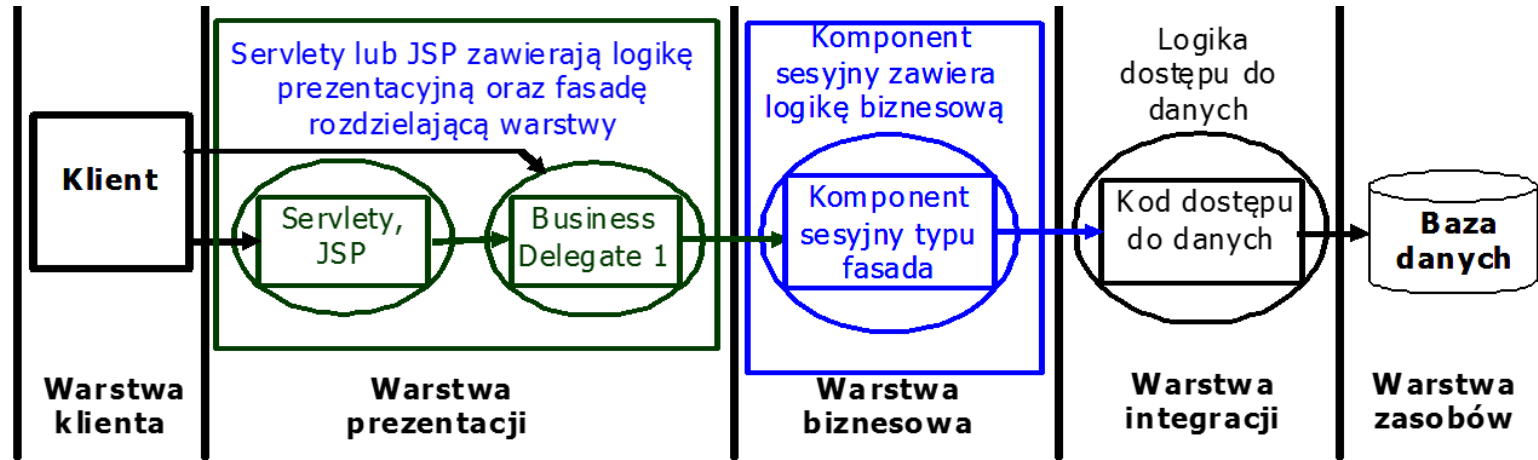
Refaktoryzacja architektury wielowarstwowej - część 1

1. Przeniesienie kodu dostępu do danych logicznie lub fizycznie bliżej rzeczywistego źródła danych ➡ **warstwa integracji**
2. Przeniesienie kodu logiki przetwarzania z 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”



Zagadnienia

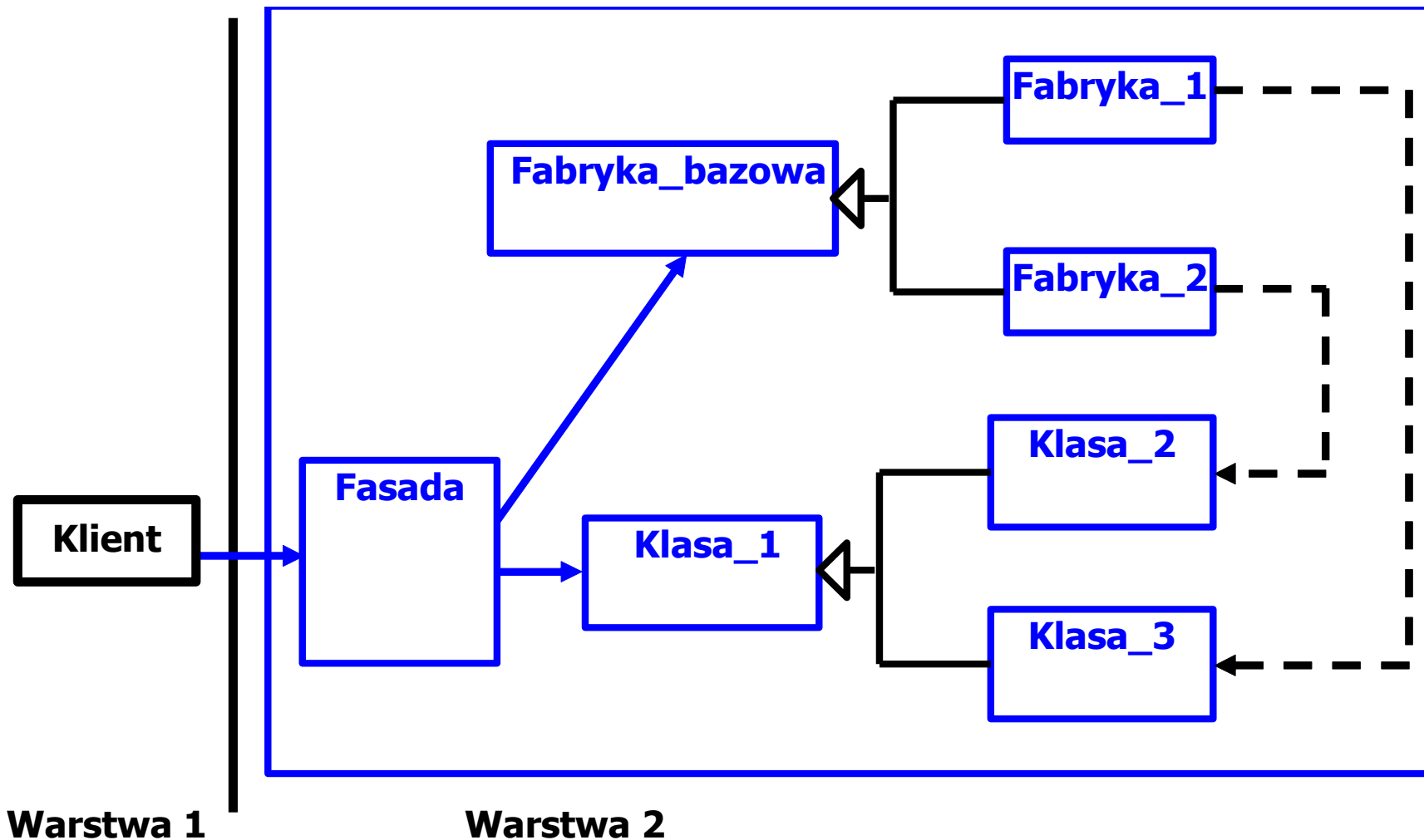
- 1. Wielowarstwowa architektura systemu informatycznego**
- 2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego**
- 3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej [6]**

Identyfikacja wzorców projektowych

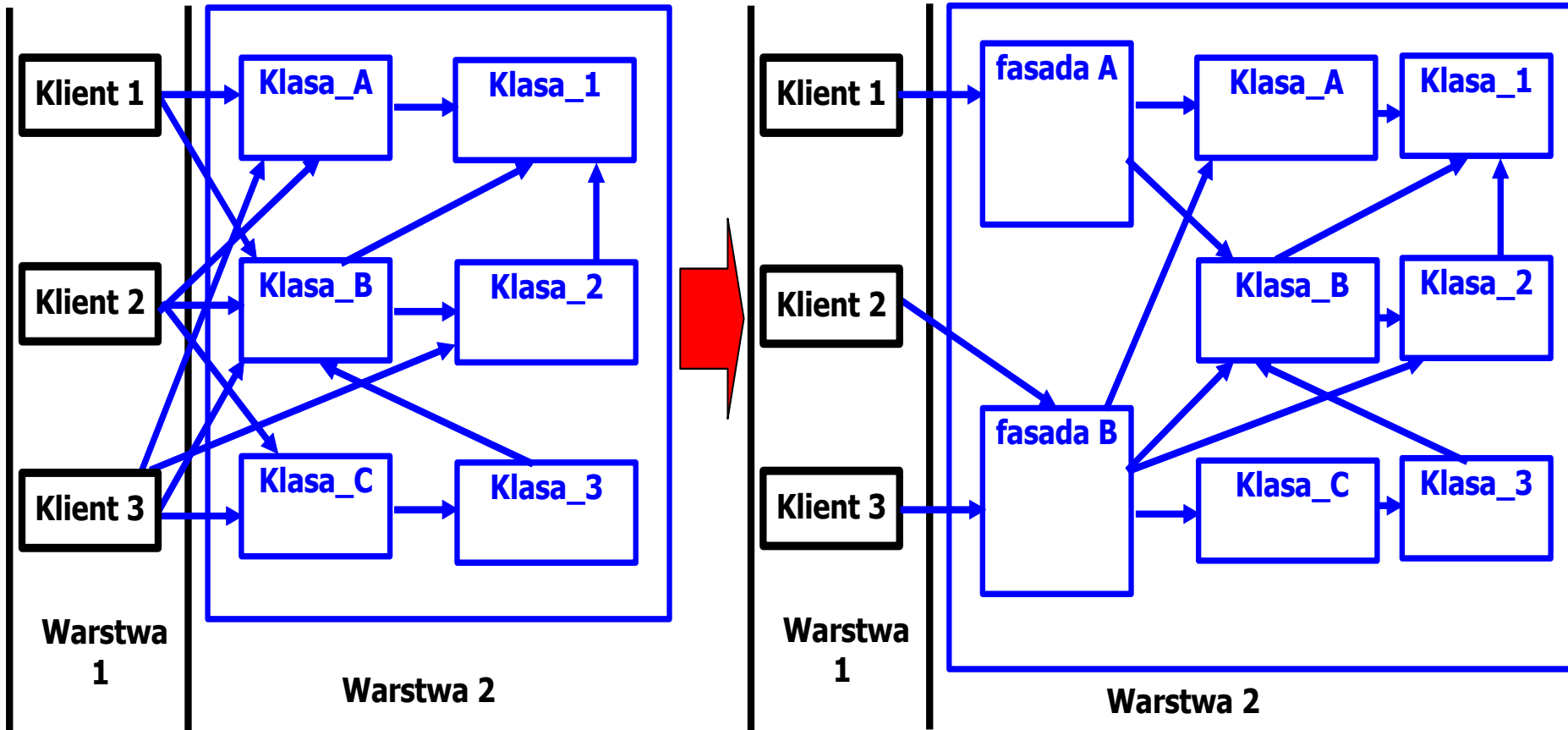
- Dobrze zbudowany system obiektowy jest pełen wzorców obiektowych
- Wzorzec to zwyczajowo przyjęte rozwiązanie typowego problemu w danym kontekście
- Strukturę wzorca przedstawia się w postaci diagramu klas
- Zachowanie się wzorca przedstawia się za pomocą diagramu sekwencji
- **Wzorce projektowe: Wzorzec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

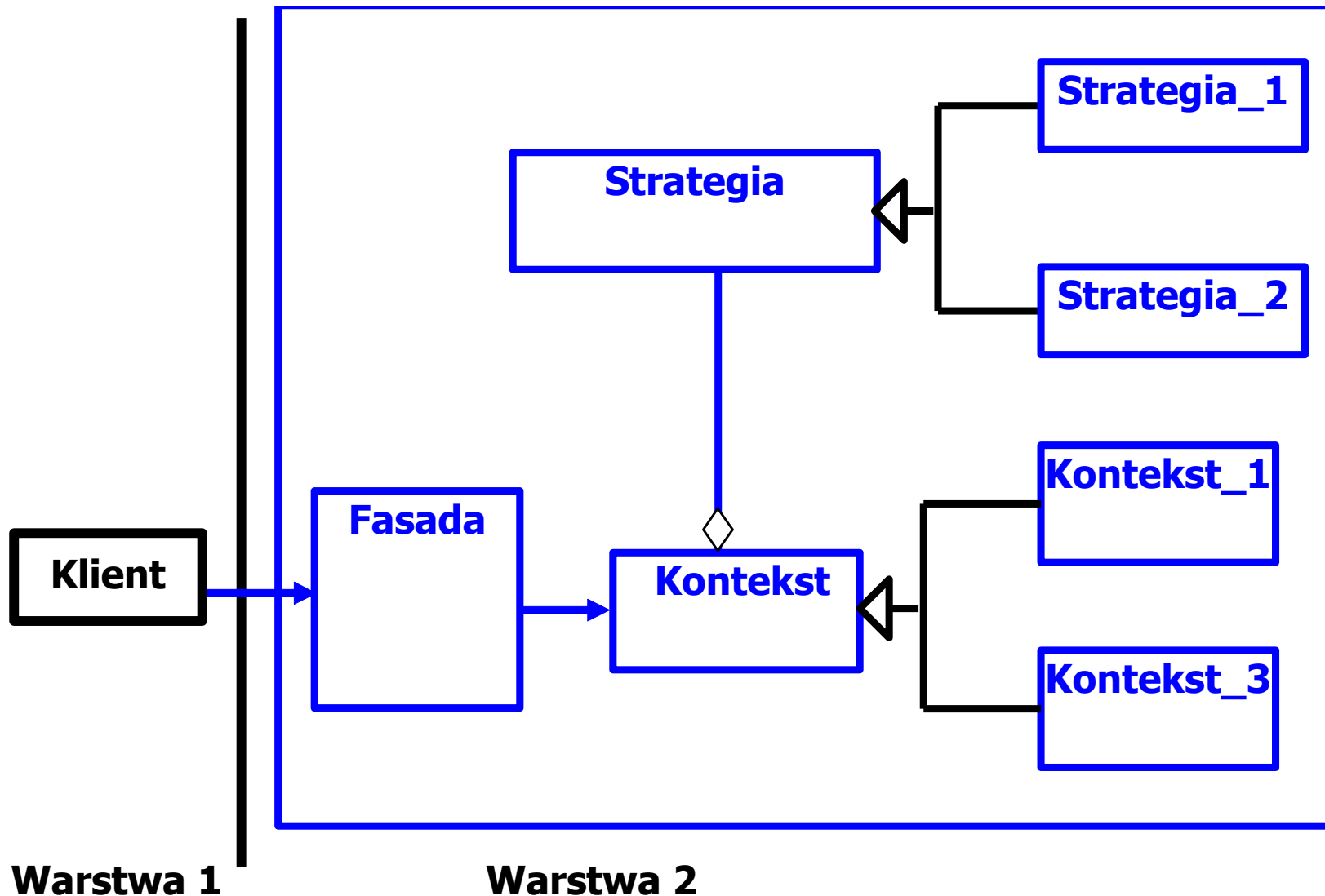
3.1. Wzorzec uniwersalny kreacyjny stosowany w każdej z warstw: **Fabryka obiektów** – oddzielenie tworzenia obiektów od zarządzania nimi i używania ich



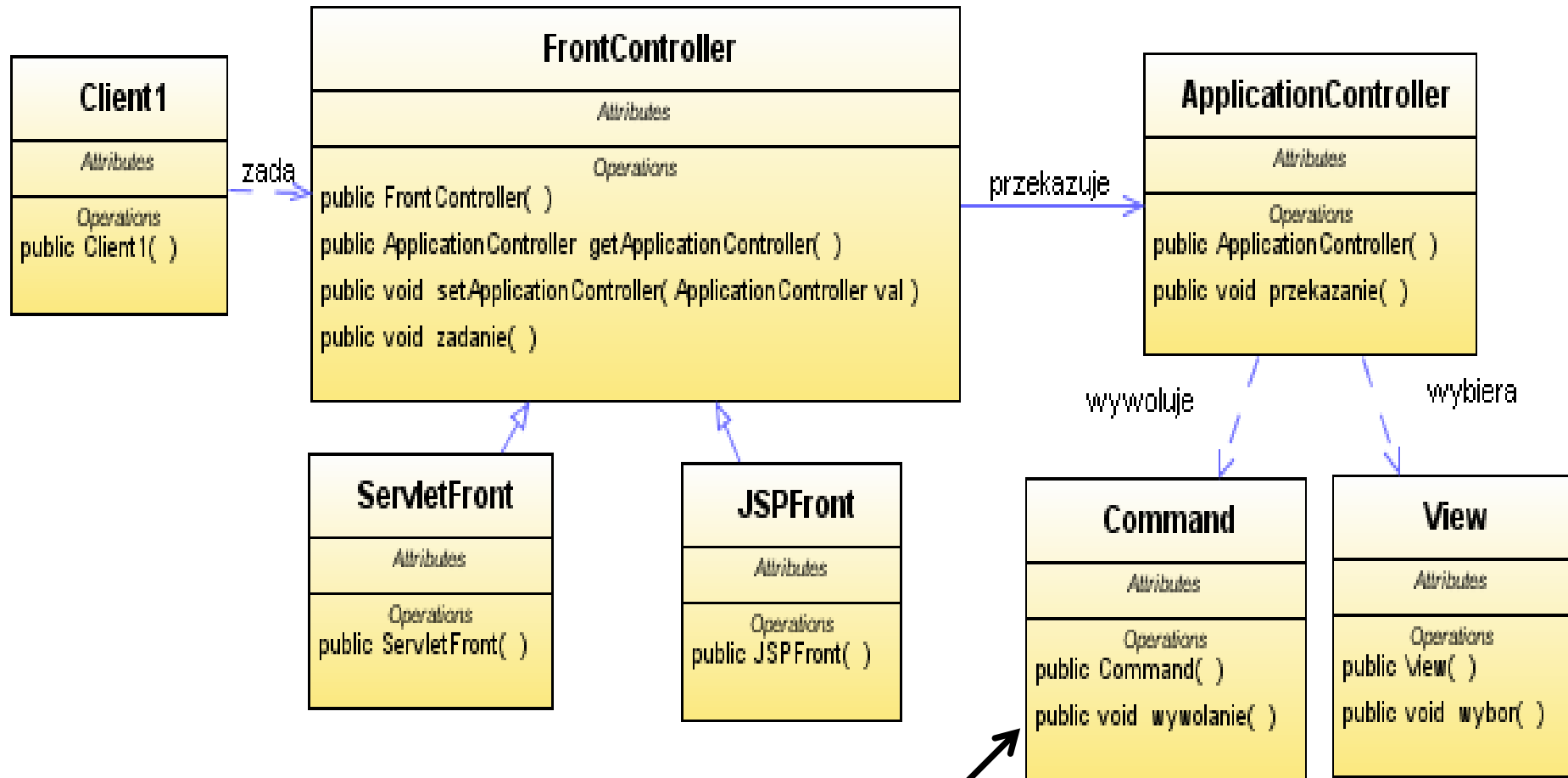
3.2. Wzorzec uniwersalny strukturalny: **Fasada** – hermetyzacja logiki biznesowej



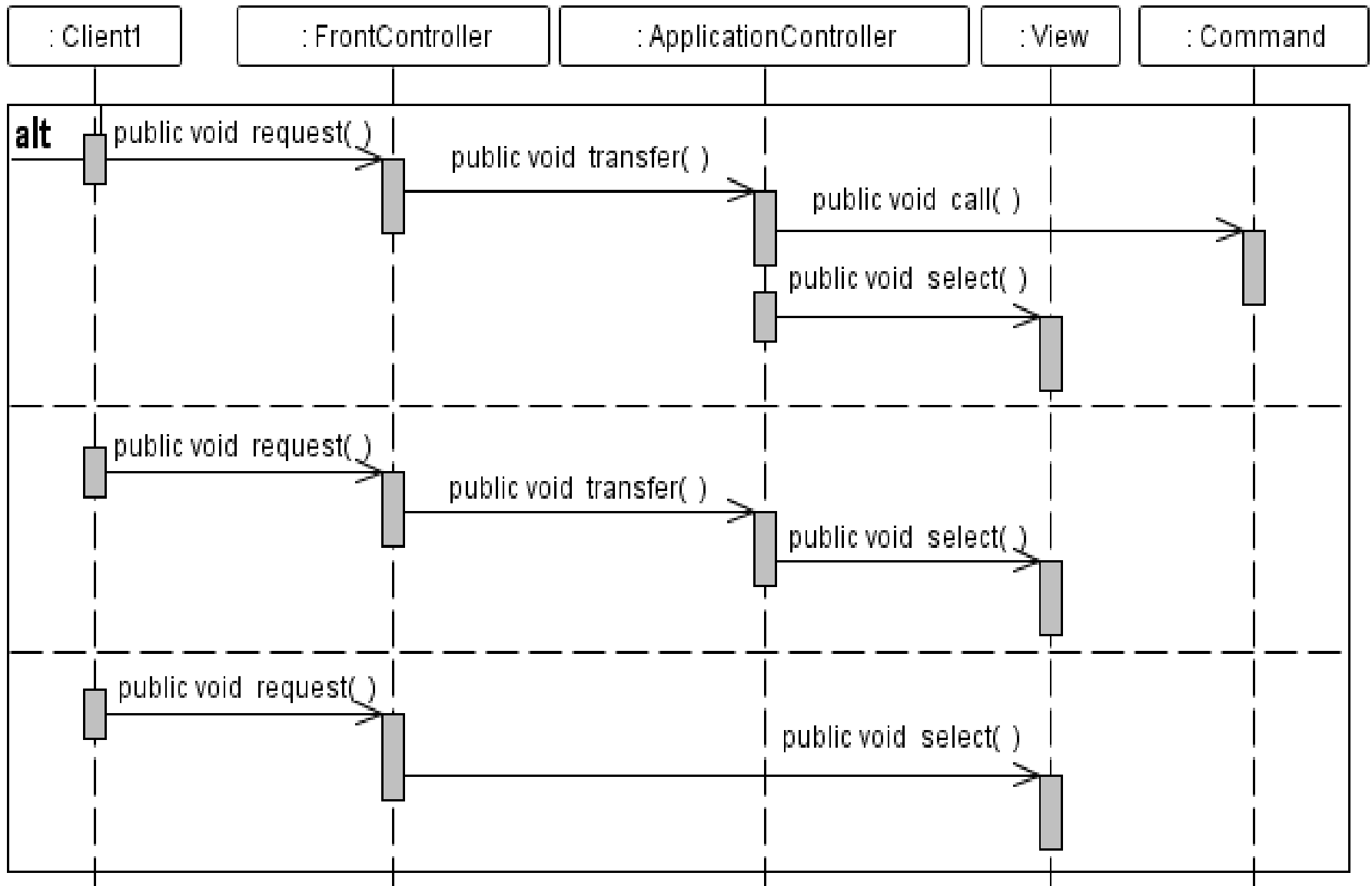
3.3. Wzorzec uniwersalny czynnościowy kreacyjny stosowany w każdej z warstw: : **Strategia** – zastosowanie polimorfizmu do wyboru algorytmu



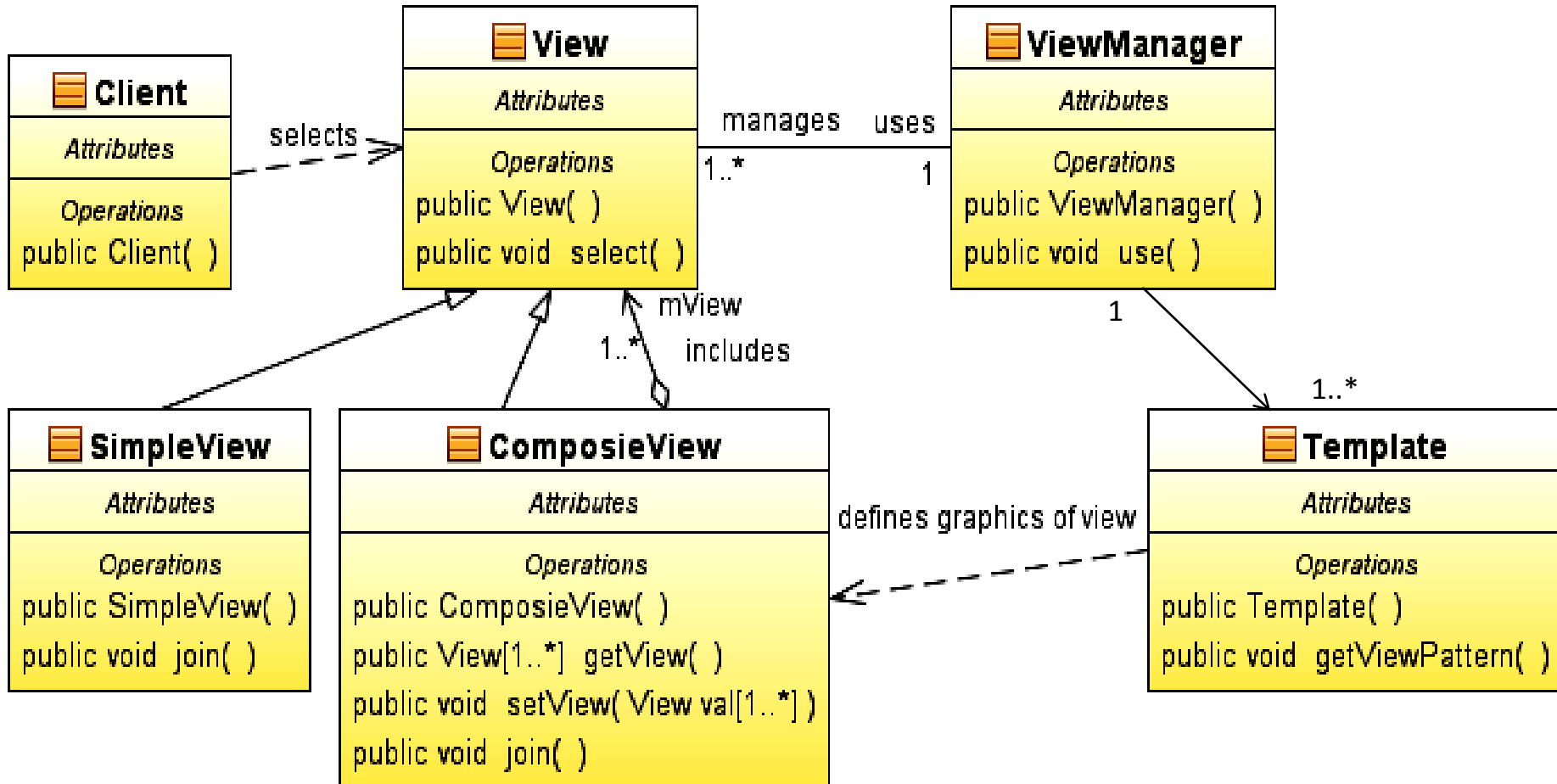
3.4. Wzorzec EE warstwy internetowej: *FrontController* – scentralizowany punkt dostępowy do obsługi żądań w warstwie internetowej



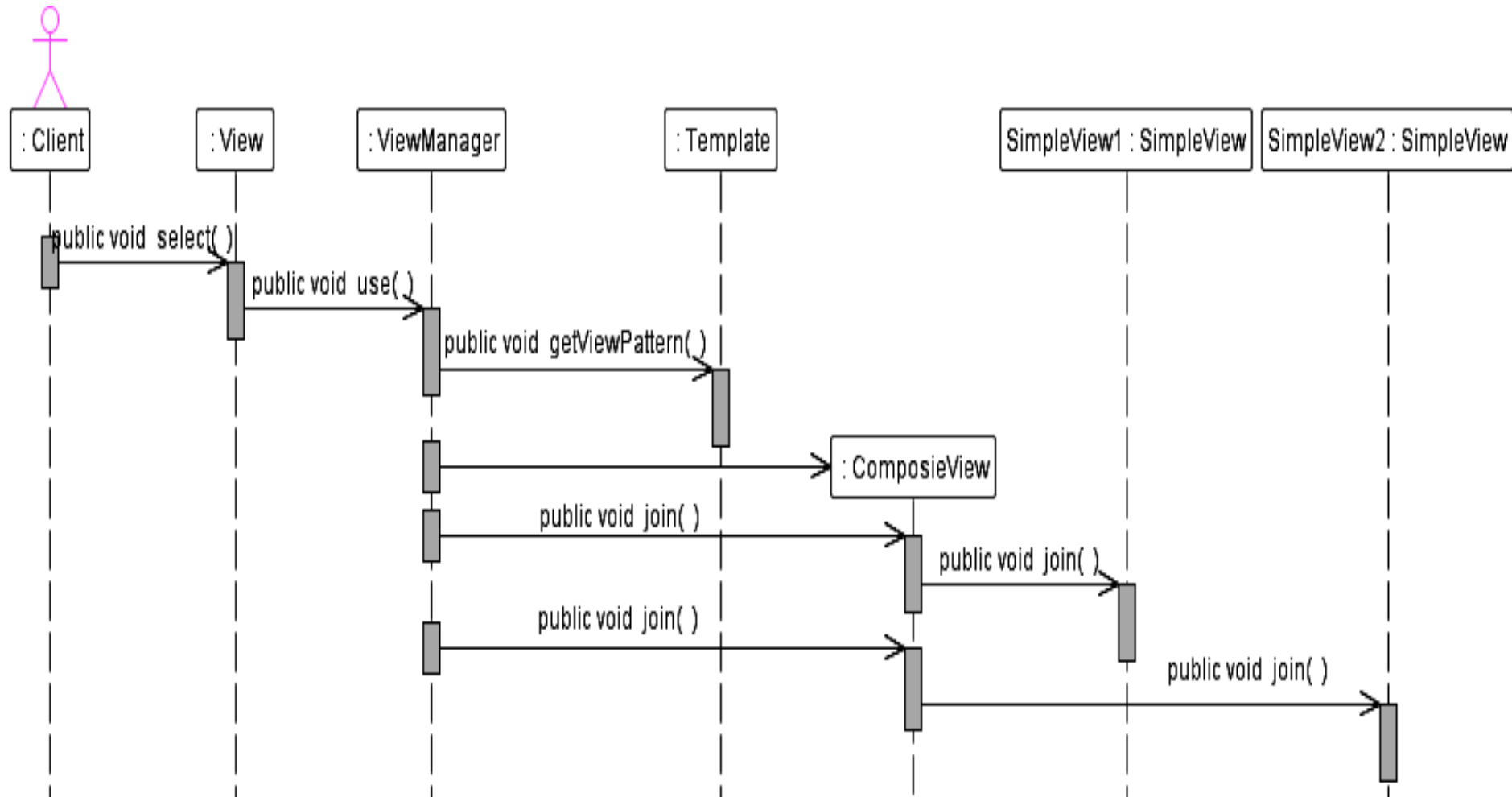
Wywołanie metod z warstwy biznesowej np. za pośrednictwem wzorca `ApplicationService`



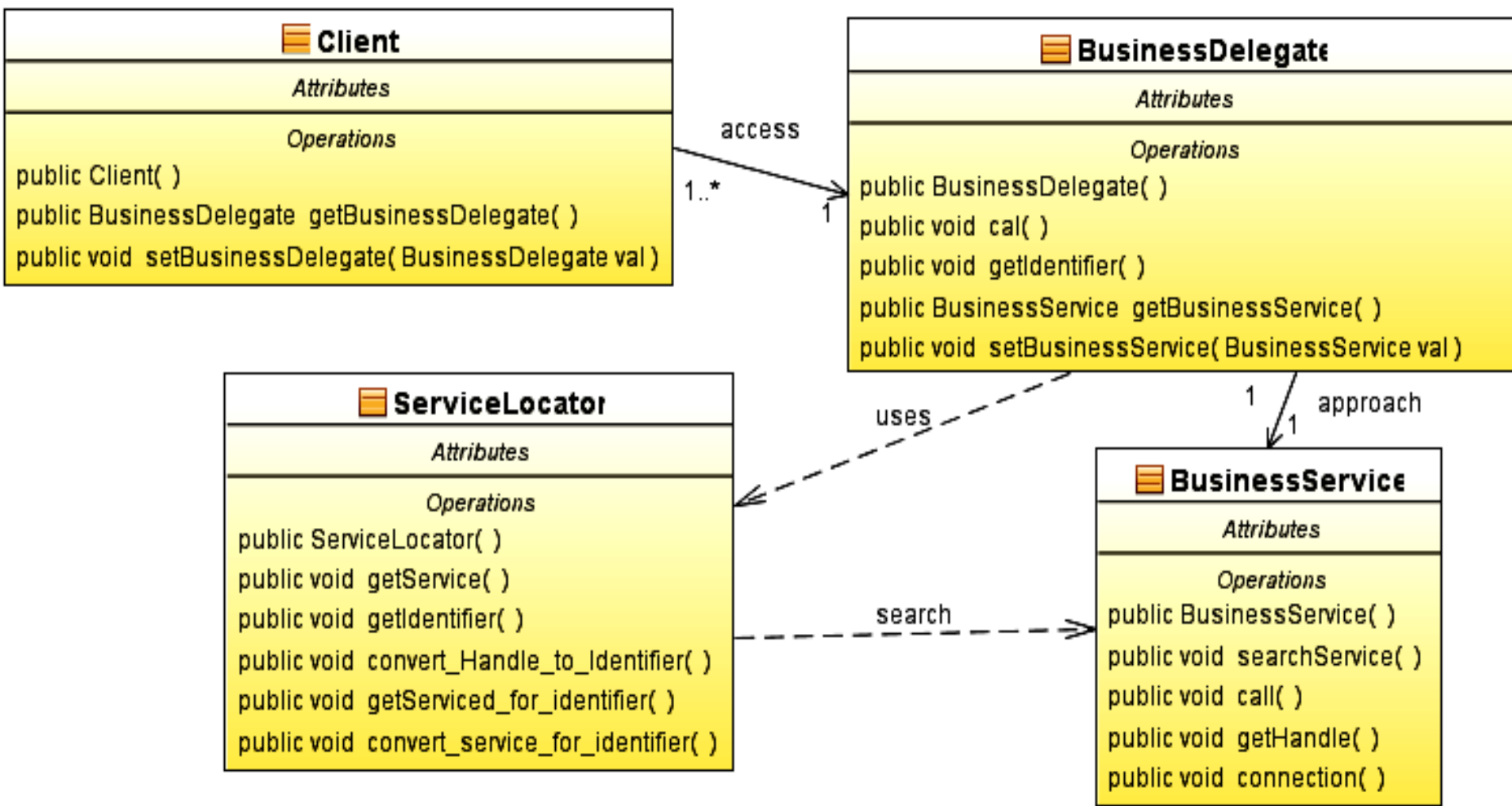
3.5. Wzorzec EE warstwy internetowej: *Composite View* - widok kompozytowy powinien mieć strukturę modułową, zbudowaną z komponentów prostych, które razem tworzą złożoną stronę są zarządzane niezależnie.



Tworzenie złożonego widoku wykonanego z wieloużywalnych komponentów, oparty na szablonie widoku

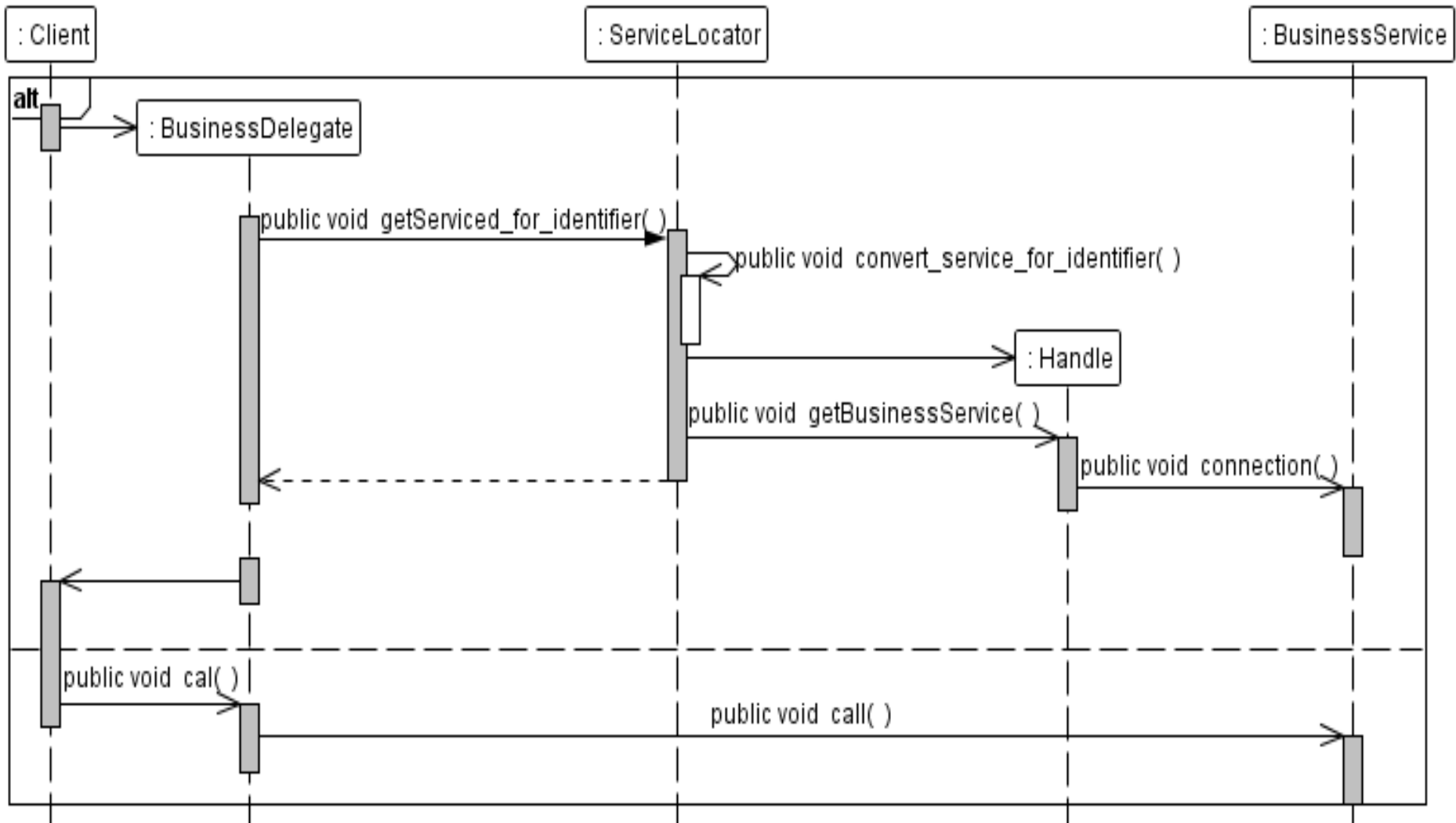


3.6. Wzorzec EE warstwy klienta: do zdalnego wywołania usług z warstwy klienta w celu ukrycia złożoności zdalnej komunikacji z komponentem usług biznesowych - *BusinessDelegate*



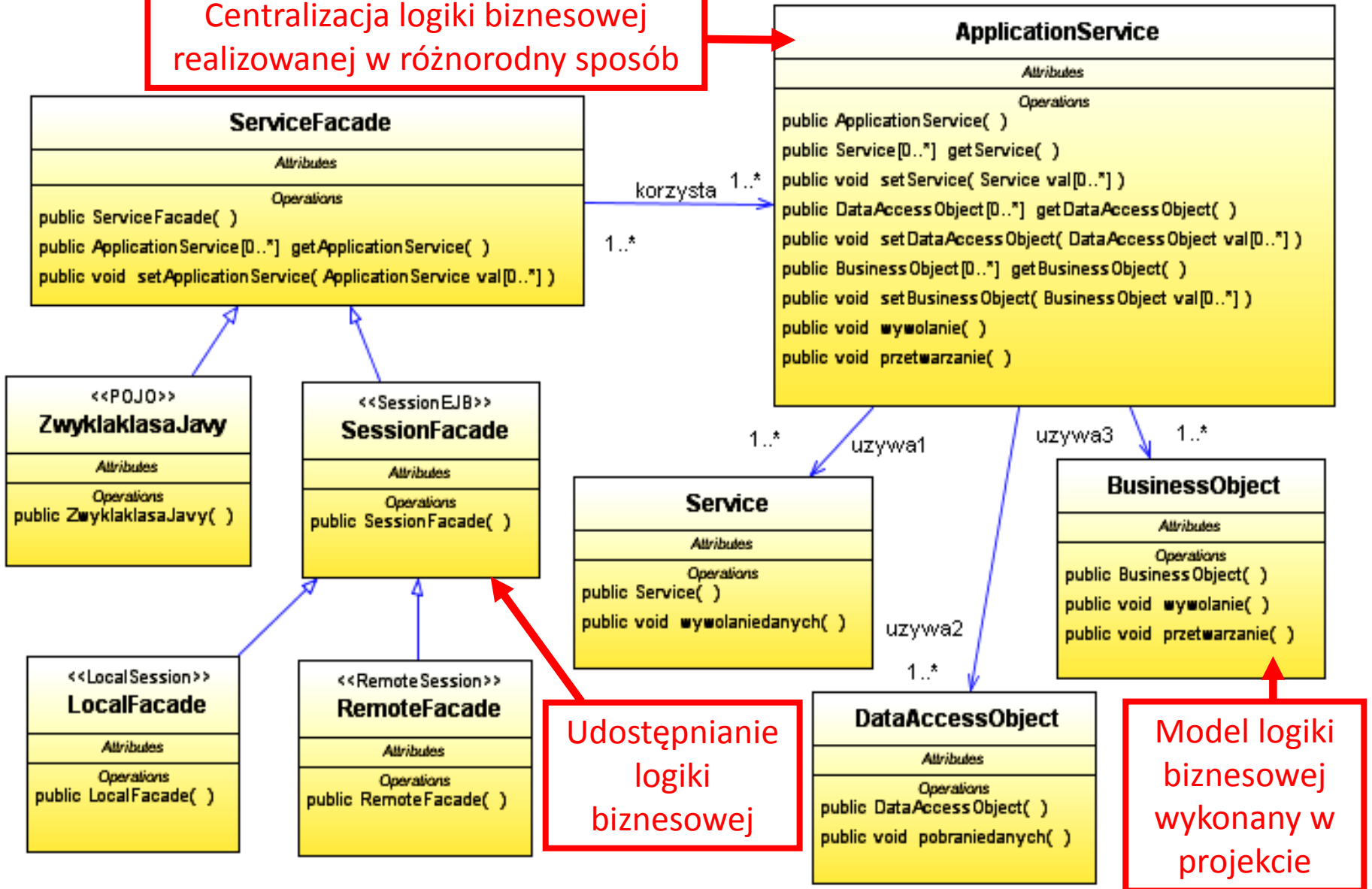
Komponenty EJB, JMS , lub część wzorca *SessionFacade*

Wywołanie usługi biznesowej



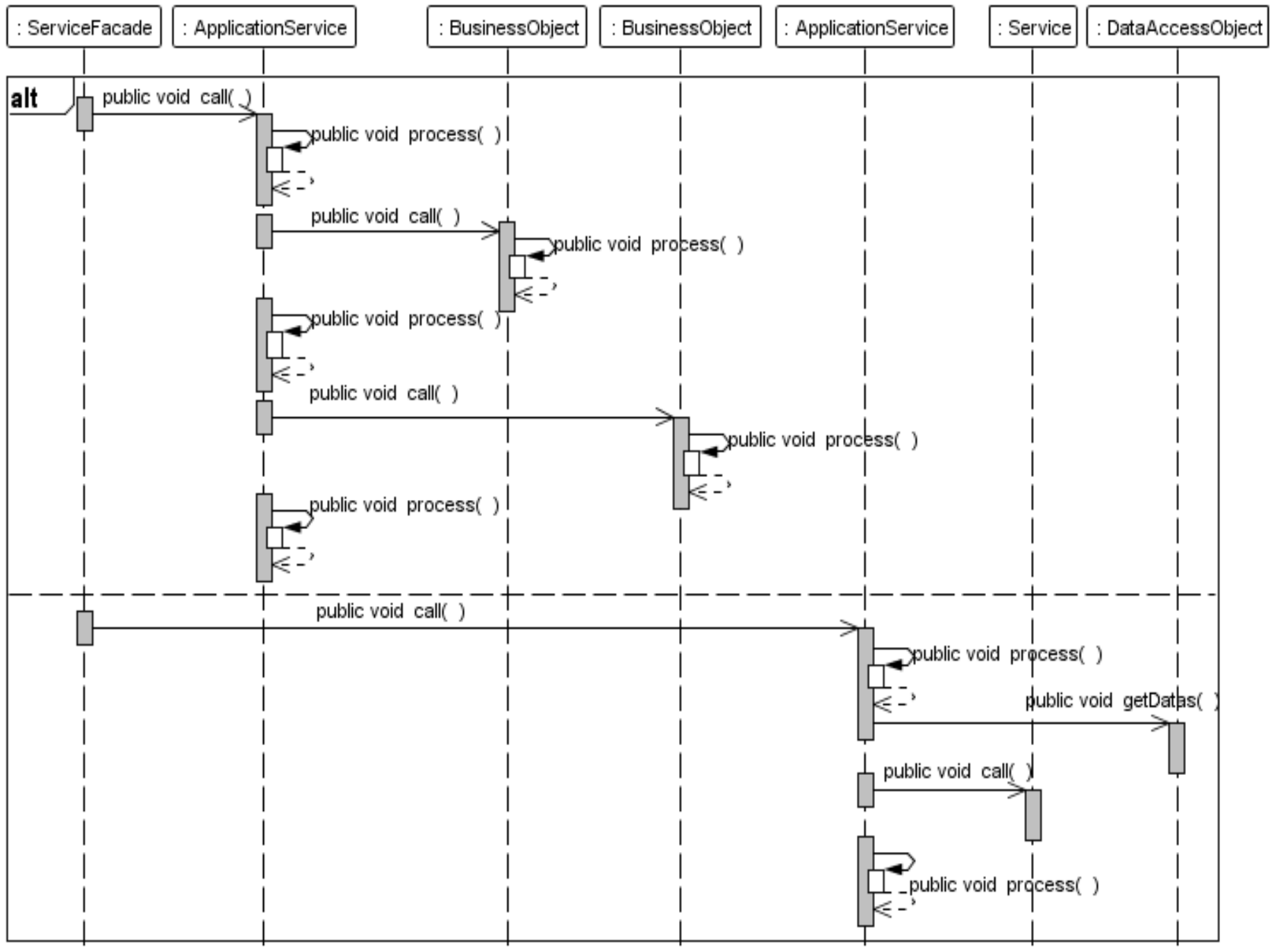
3.7. Wzorce EE warstwy biznesowej: *SessionFacade*, *ApplicationService* – udostępnianie i centralizacja logiki biznesowej kilku komponentów i usług biznesowych

Centralizacja logiki biznesowej realizowanej w różnorodny sposób

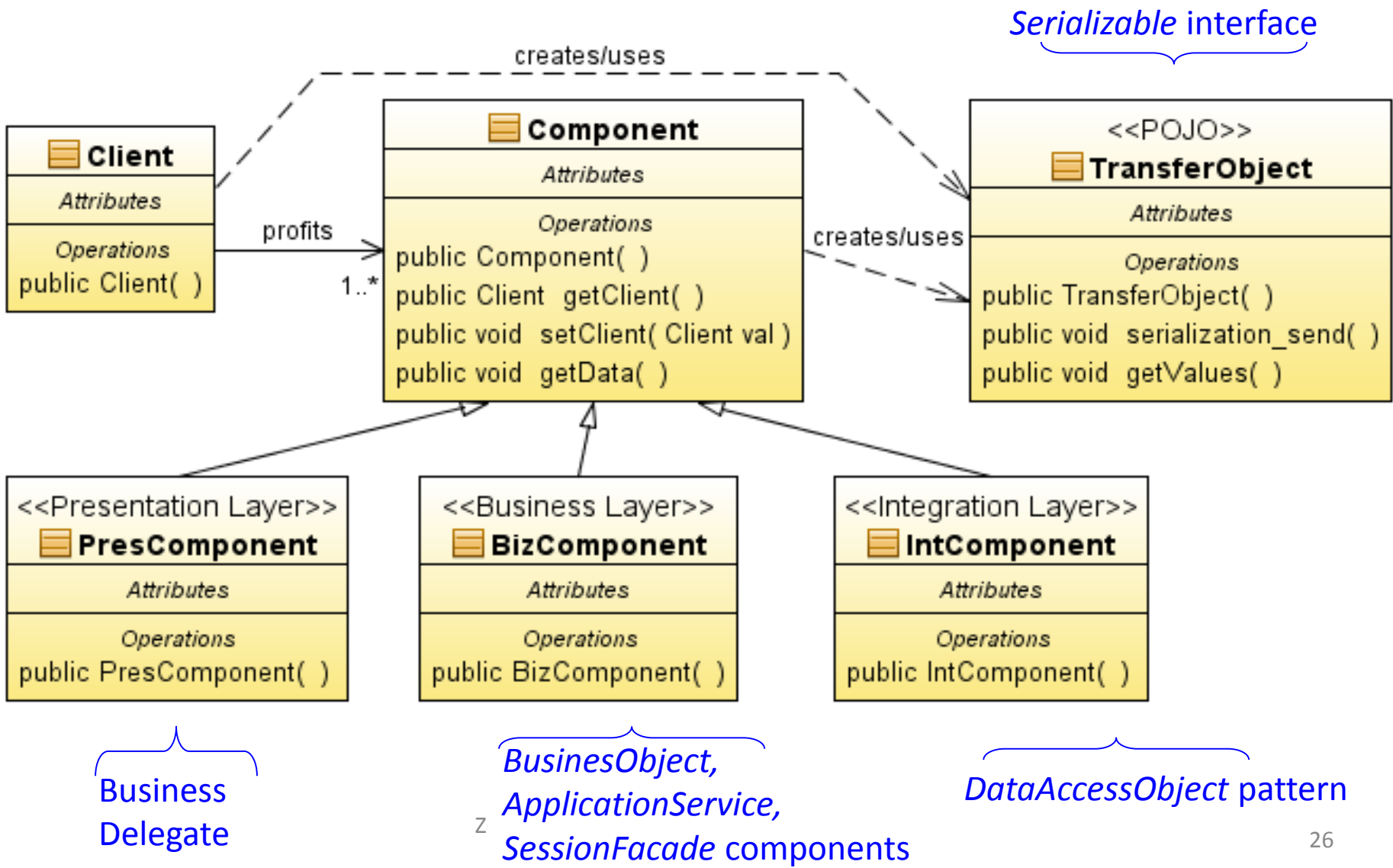


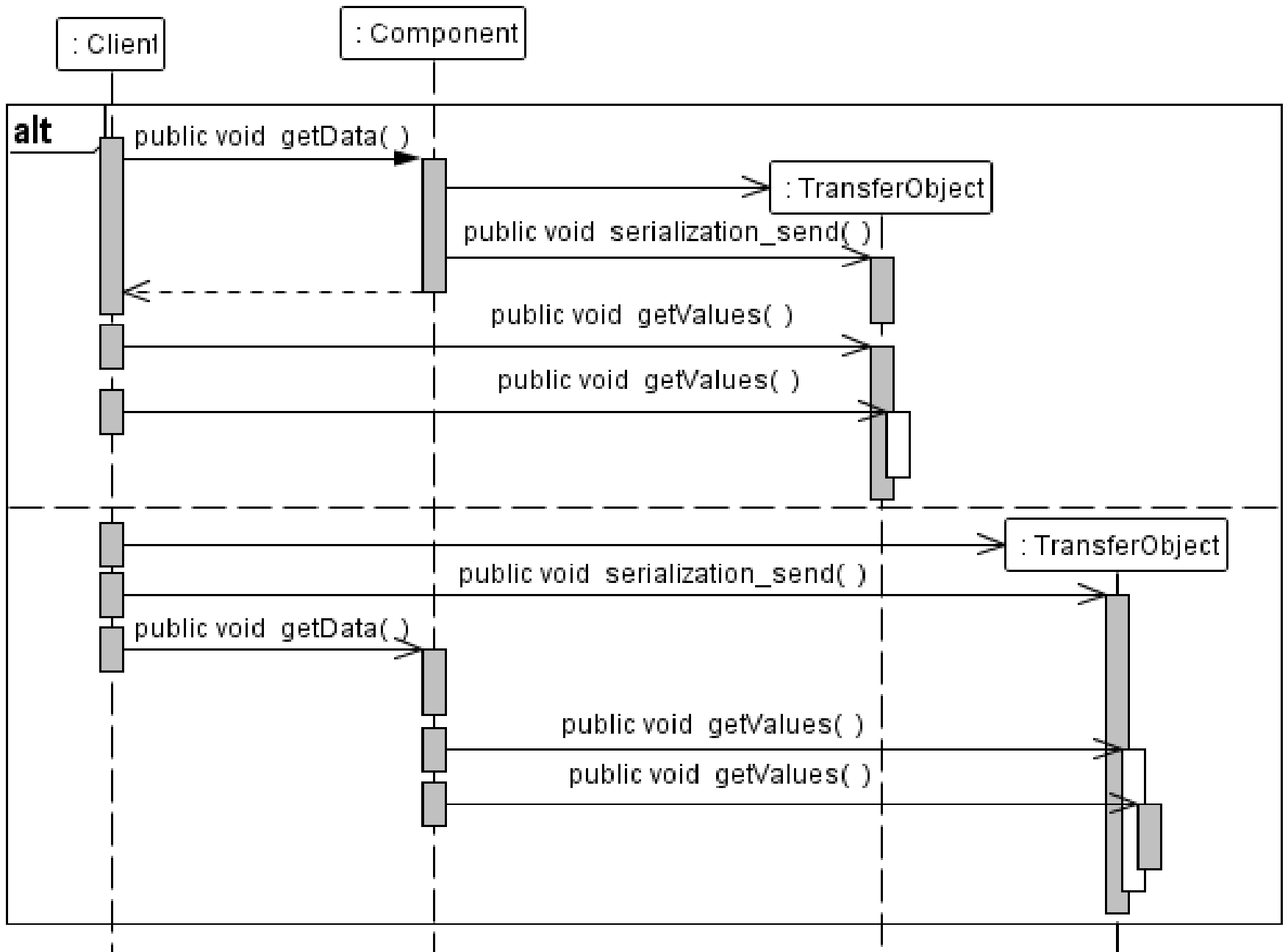
Udostępnianie logiki biznesowej

Model logiki biznesowej wykonany w projekcie

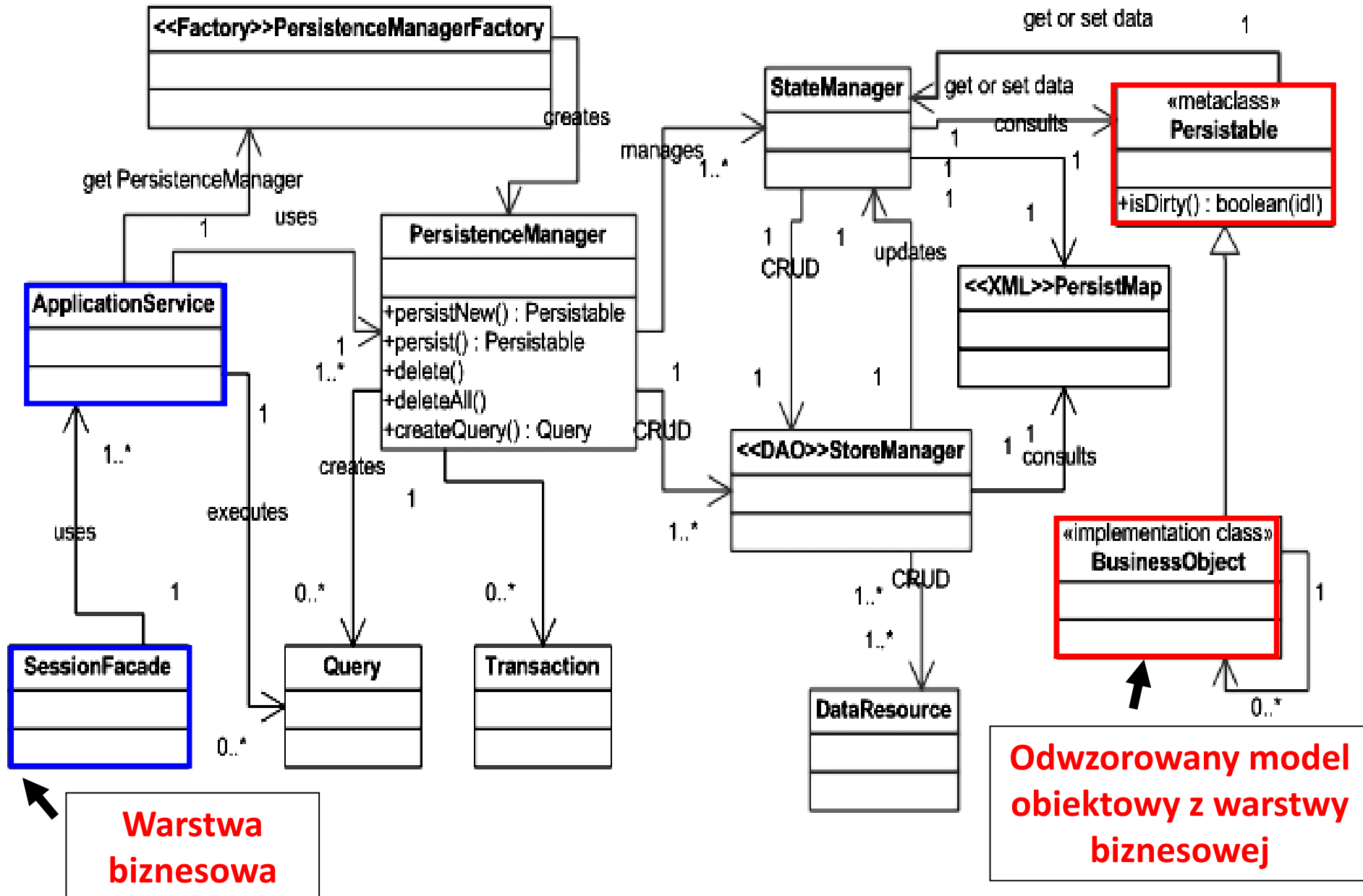


3.8. Wzorce EE warstwy biznesowej: *TransferObject* - przesyłanie danych między warstwami aplikacji (zmniejszanie ruchu w sieci poprzez zmniejszanie liczby połączeń zdalnych lub zwiększanie wydajności)

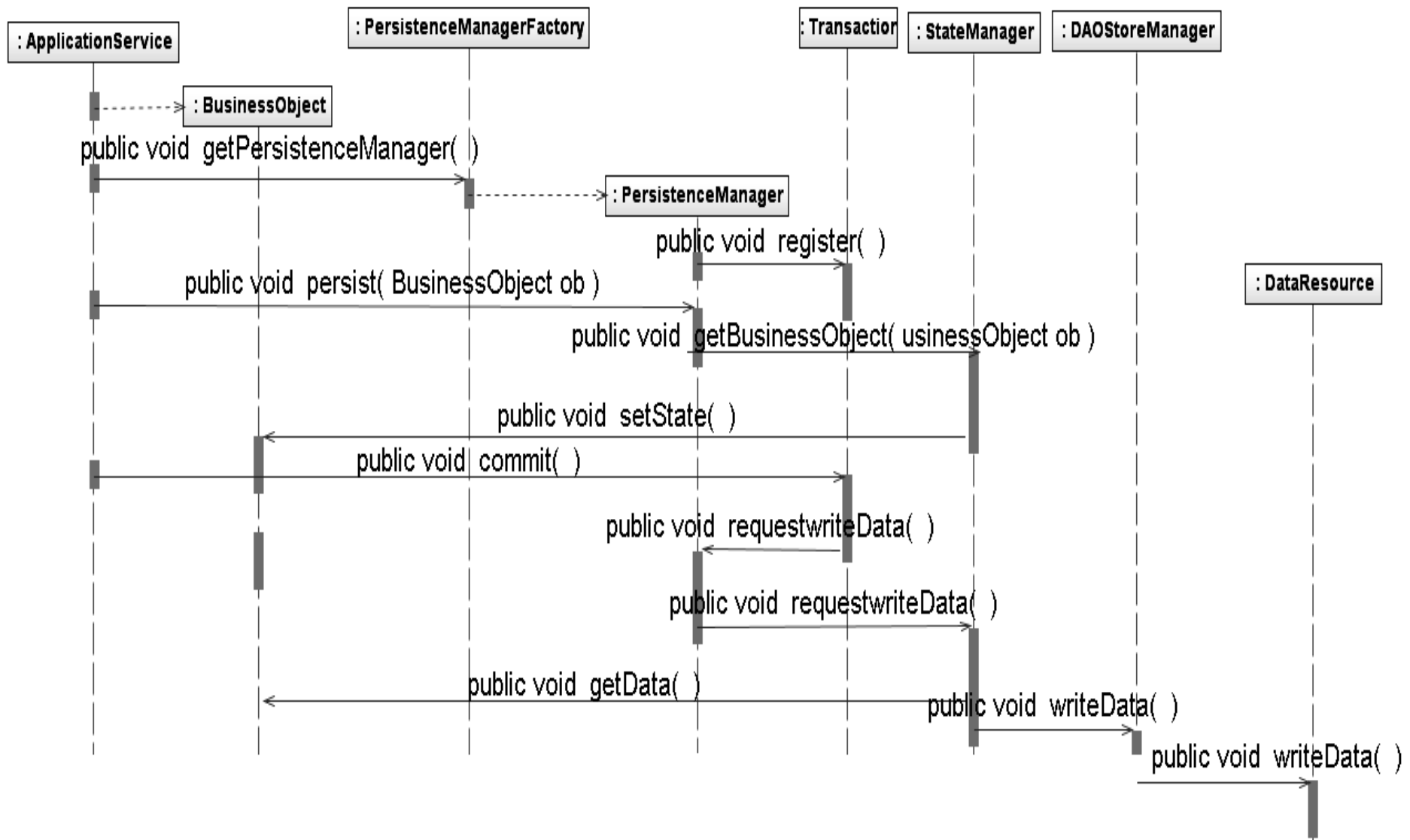




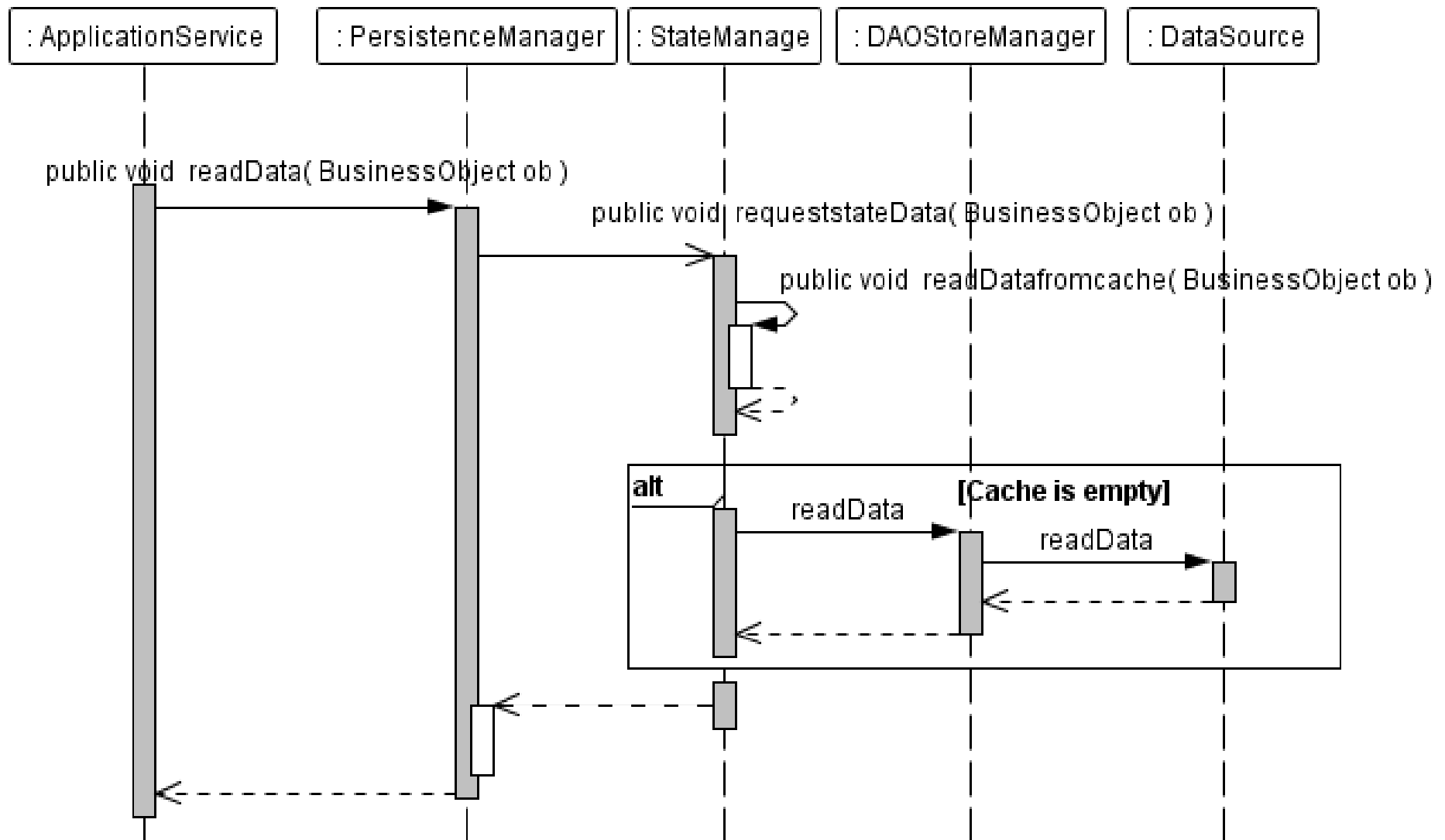
3.9. Wzorzec EE warstwy integracji: **DomainStore (ORM)** – oddzielenie mechanizmów trwałości od modelu obiektowego



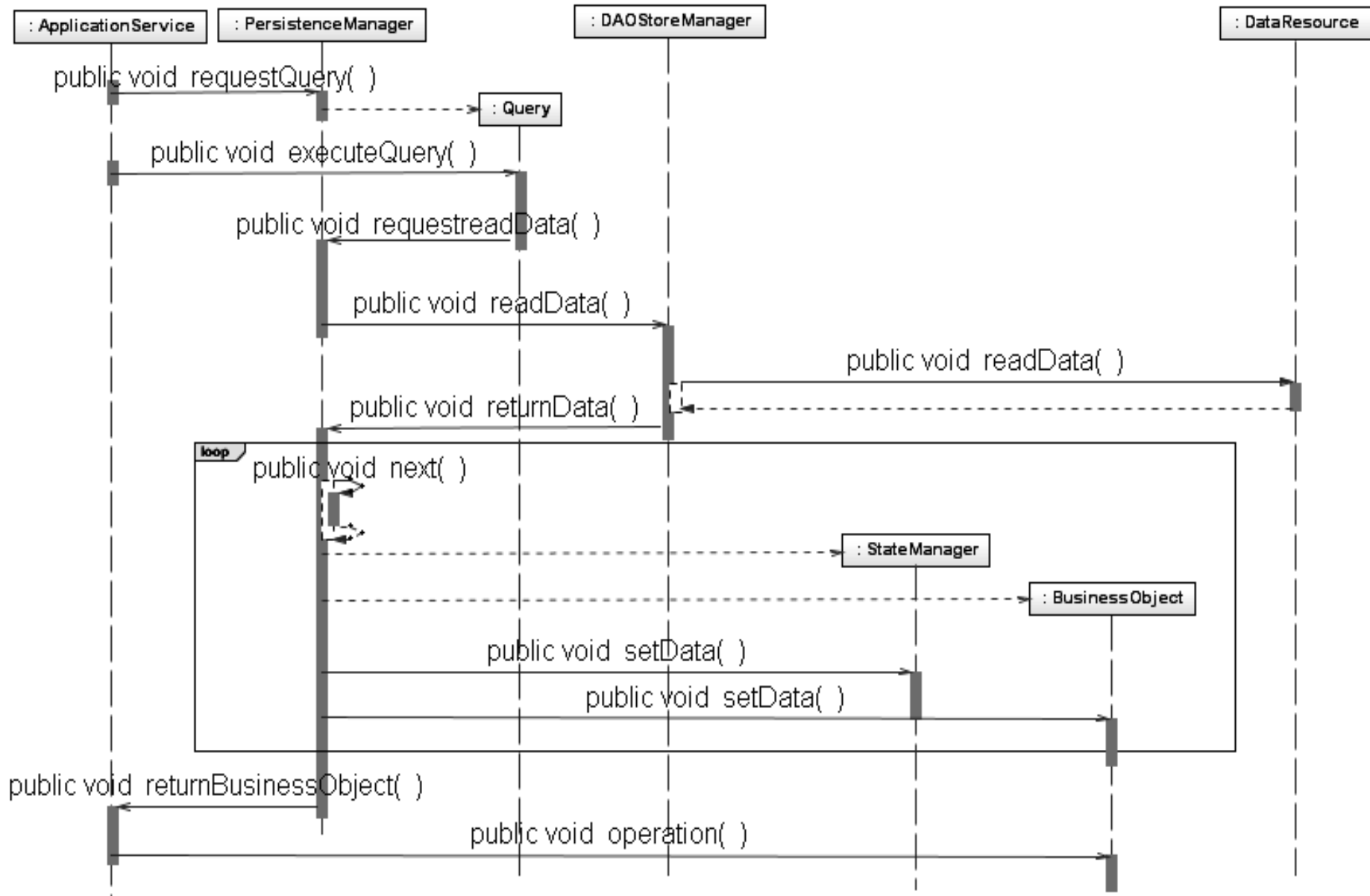
Tworzenie i utrwalanie obiektów (*Business Object*)



Zapełnianie buforów



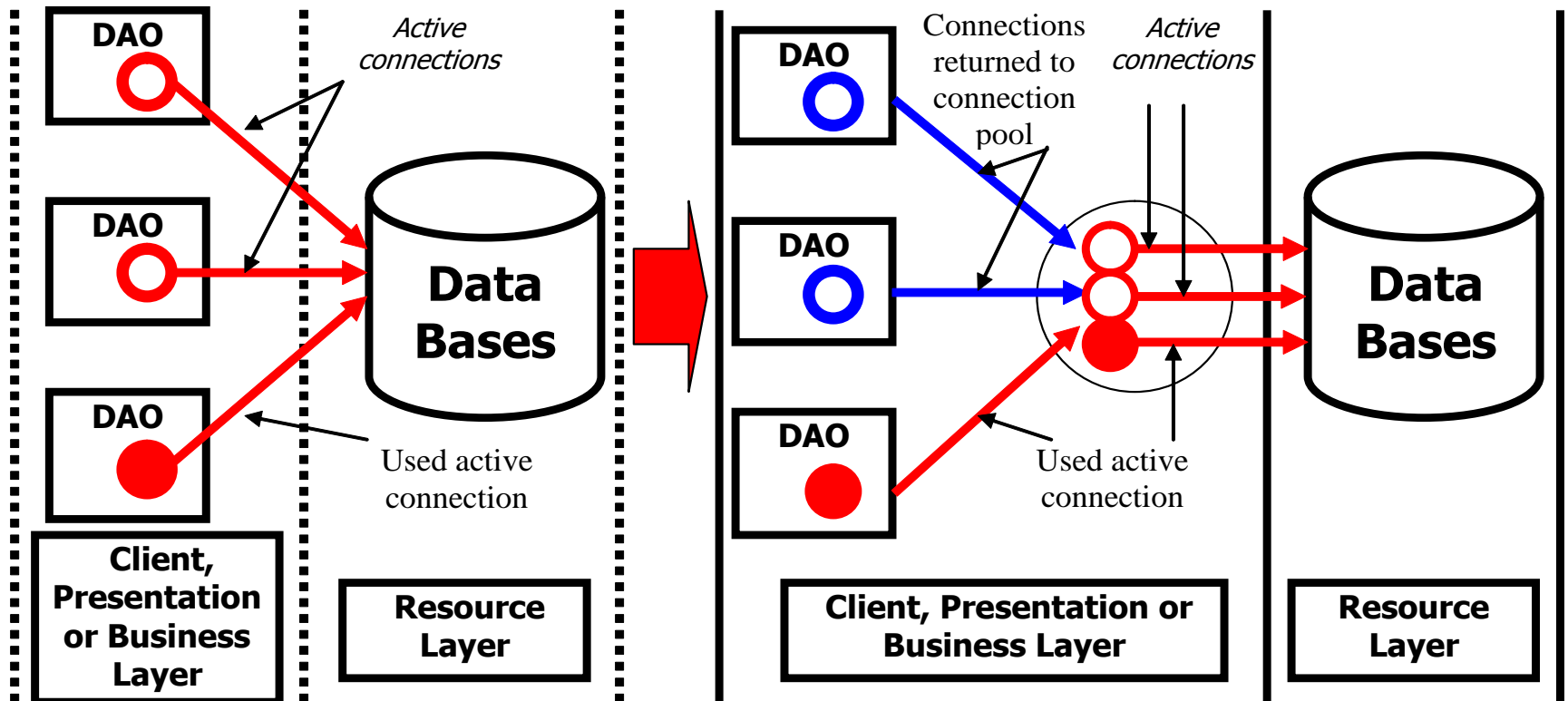
Tworzenie i wykonanie zapytanie (*Query*)



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



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

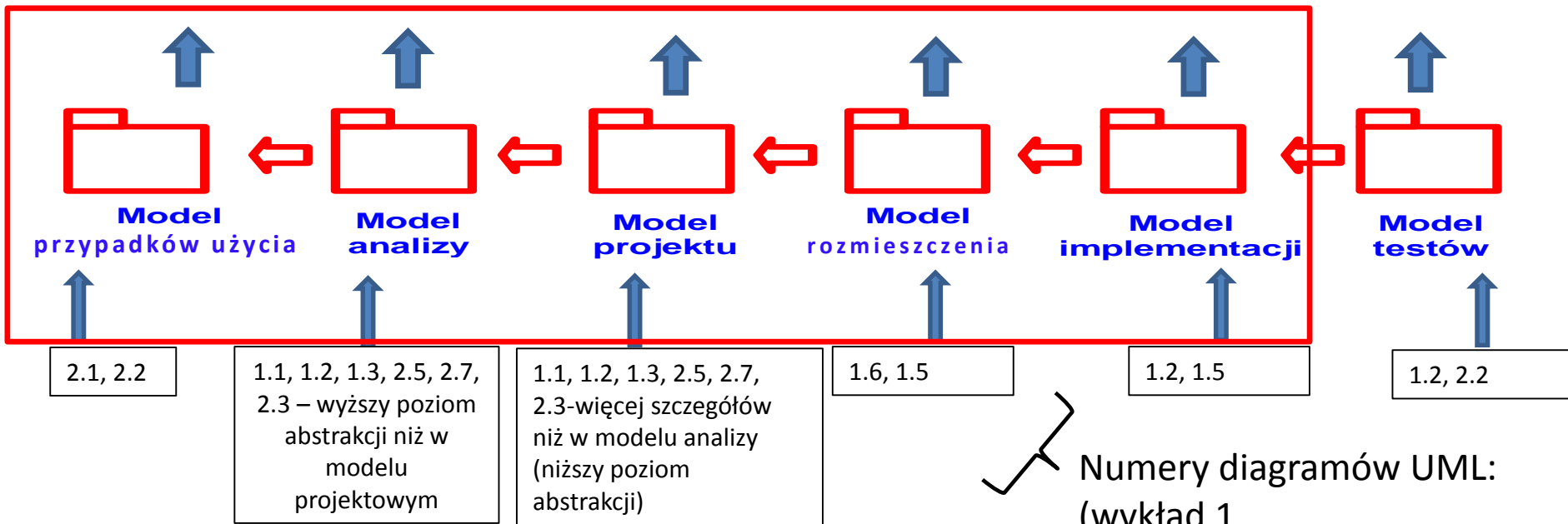
Requests no.	Jdbc[ms] (DAO)	Jndi[ms] (DAO and pool of connections)	Orm[ms] (Domain Store)
			lazy
20	8 431	5 136	2 627
100	49 356	19 311	2 445

Zagadnienia

- 1. Wielowarstwowa architektura systemu informatycznego**
- 2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego**
- 3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej**
- 4. Przykład modelowania i projektowania części warstwy biznesowej z obiektami typu POJO. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.**

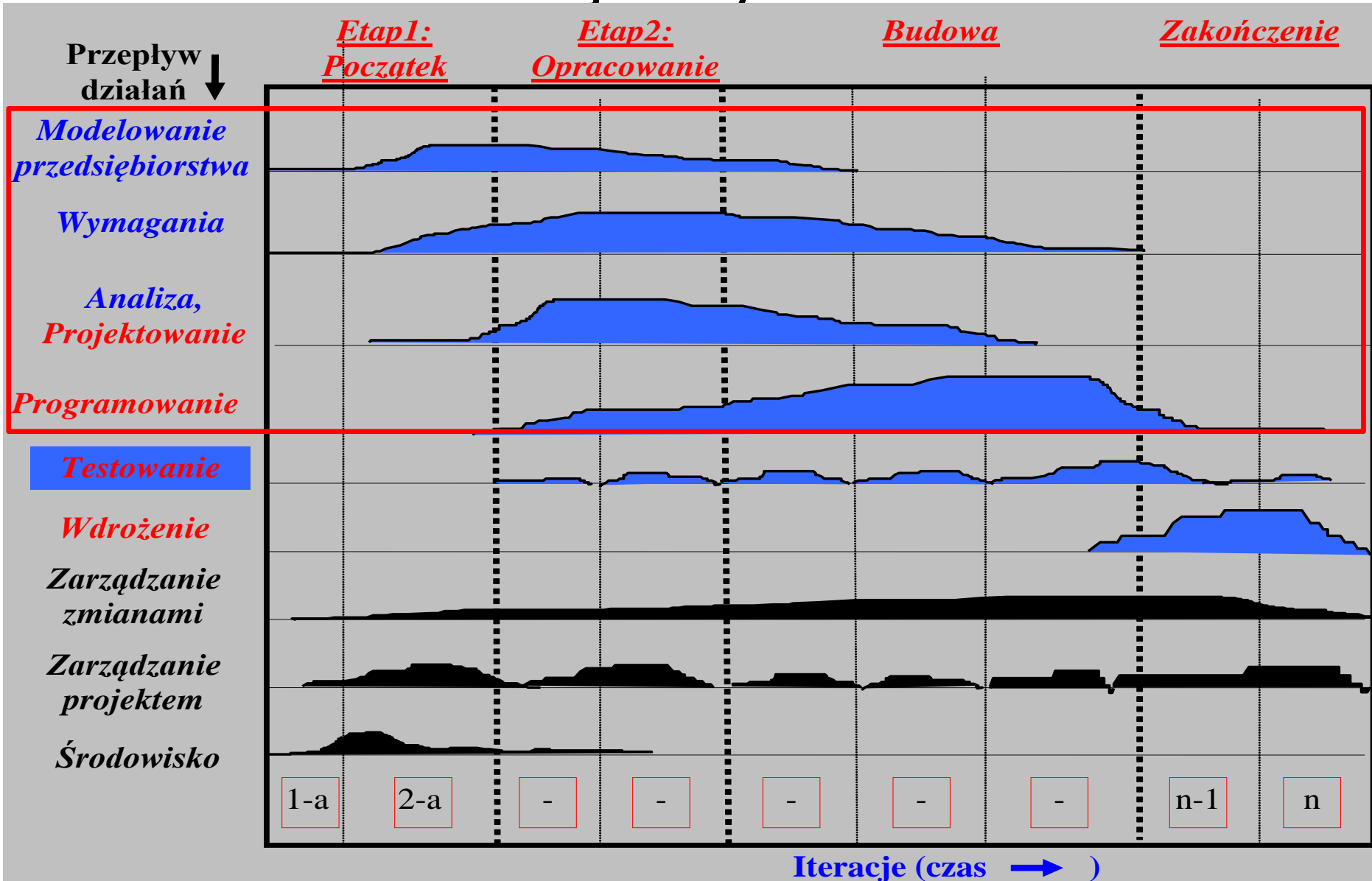
Produkt - diagramy UML – modele, proces (wykład 1)

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> • analiza (model konceptualny: diagram przypadków użycia, diagram klas, diagramy sekwencji,) • testy modelu 	<ul style="list-style-type: none"> • projektowanie (model projektowy: architektura sprzętu i oprogramowania; dostęp użytkownika; przechowywanie danych) • testy projektu 	<ul style="list-style-type: none"> • programowanie, wdrażanie (specyfikacja programu : deklaracje, definicje; dodatkowe struktury danych: struktury „pojemnikowe”, pliki, bazy danych) • testy oprogramowania • wdrażanie • testy wdrażania



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

- slajd 22 wyklad 1



System informacyjny „Biblioteka”

- I. Opis biznesowy „świata rzeczywistego”
- II. Sformułowanie wymagań funkcjonalnych i niefunkcjonalnych aplikacji
- III. Model analizy aplikacji oparty na diagramie przypadków użycia
- IV. Model projektowy i implementacja warstwy biznesowej 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 nośniku, 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 нефunkcjonalnych

Lista wymagań funkcjonalnych

1. System zawiera katalog tytułów
2. System zawiera dwa typy egzemplarzy do wypożyczenia: książki i kasety z nagraniami dźwiękowymi książek.
3. Każdy egzemplarz zawiera tytuł, nazwisko autora, ISBN, wydawnictwo, jeśli jest to książka oraz dodatkowo nazwisko aktora, jeżeli jest to nagranie dźwiękowe.
4. Może wystąpić wiele egzemplarzy książek oraz kaset z tymi samymi tytułami. Każdy egzemplarz, zarówno książka i kaset, 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 kaset a oraz numer egzemplarza.
6. Zarówno egzemplarze typu książka lub kaset, mogą być przeznaczane do wypożyczenia na okres umowy oraz na okres ściśle określony.

Lista wymagań нефunkcjonalnych

1. Wstawianie danych o tytułach i egzemplarzach może odbywać się tylko przez uprawnione osoby
2. Wyszukiwanie informacji powinno 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

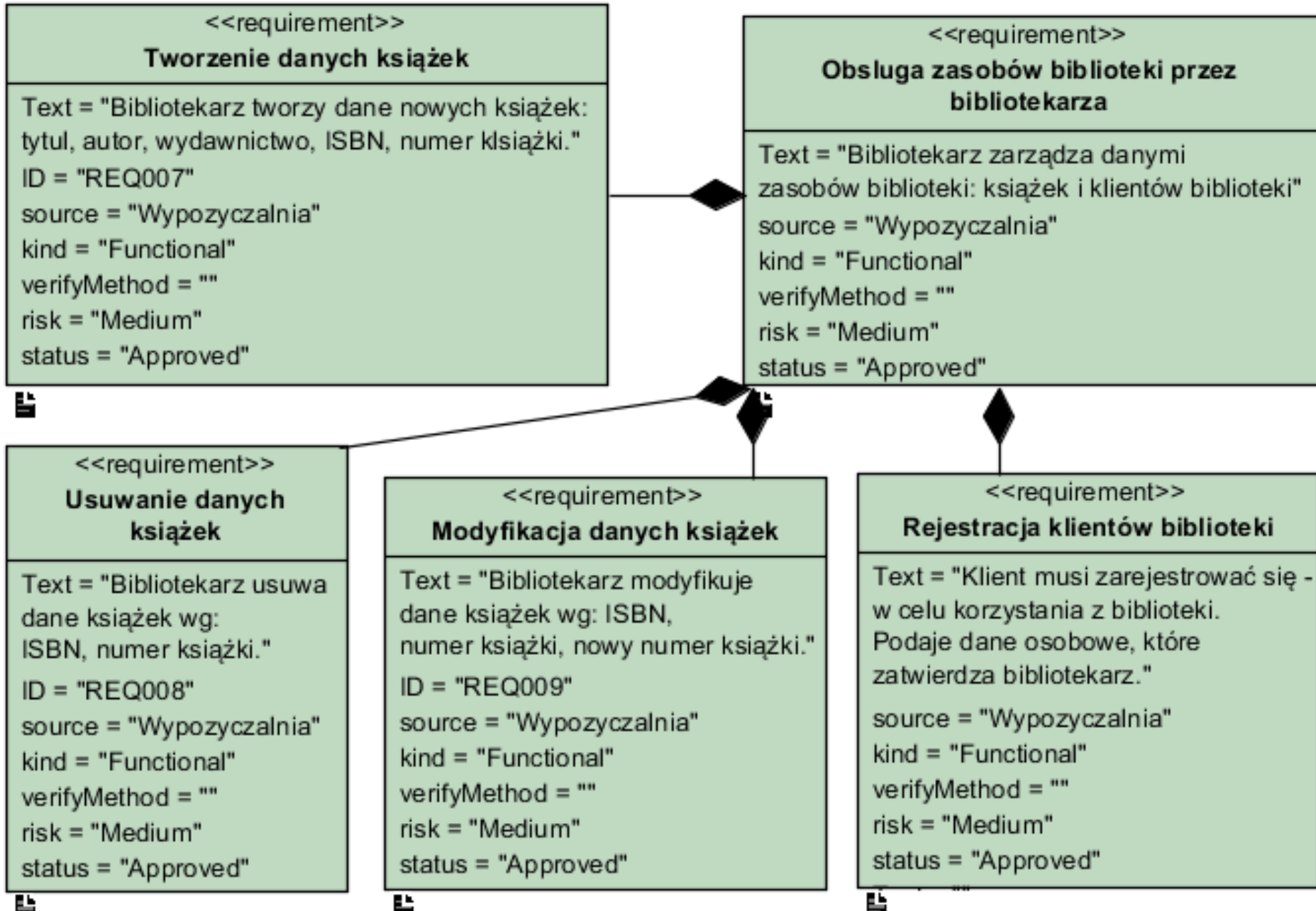


Diagram wymagań funkcjonalnych (cd)

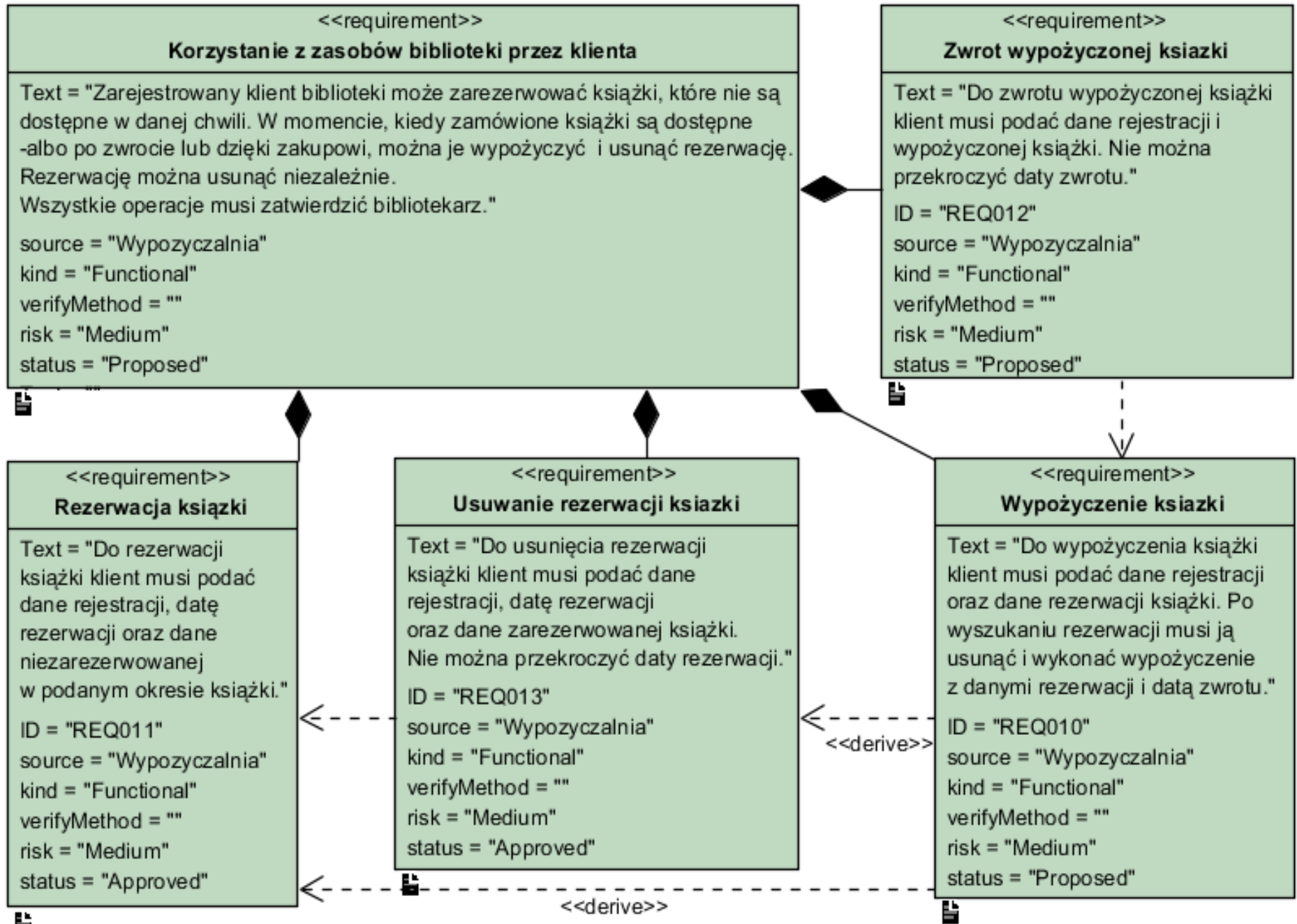
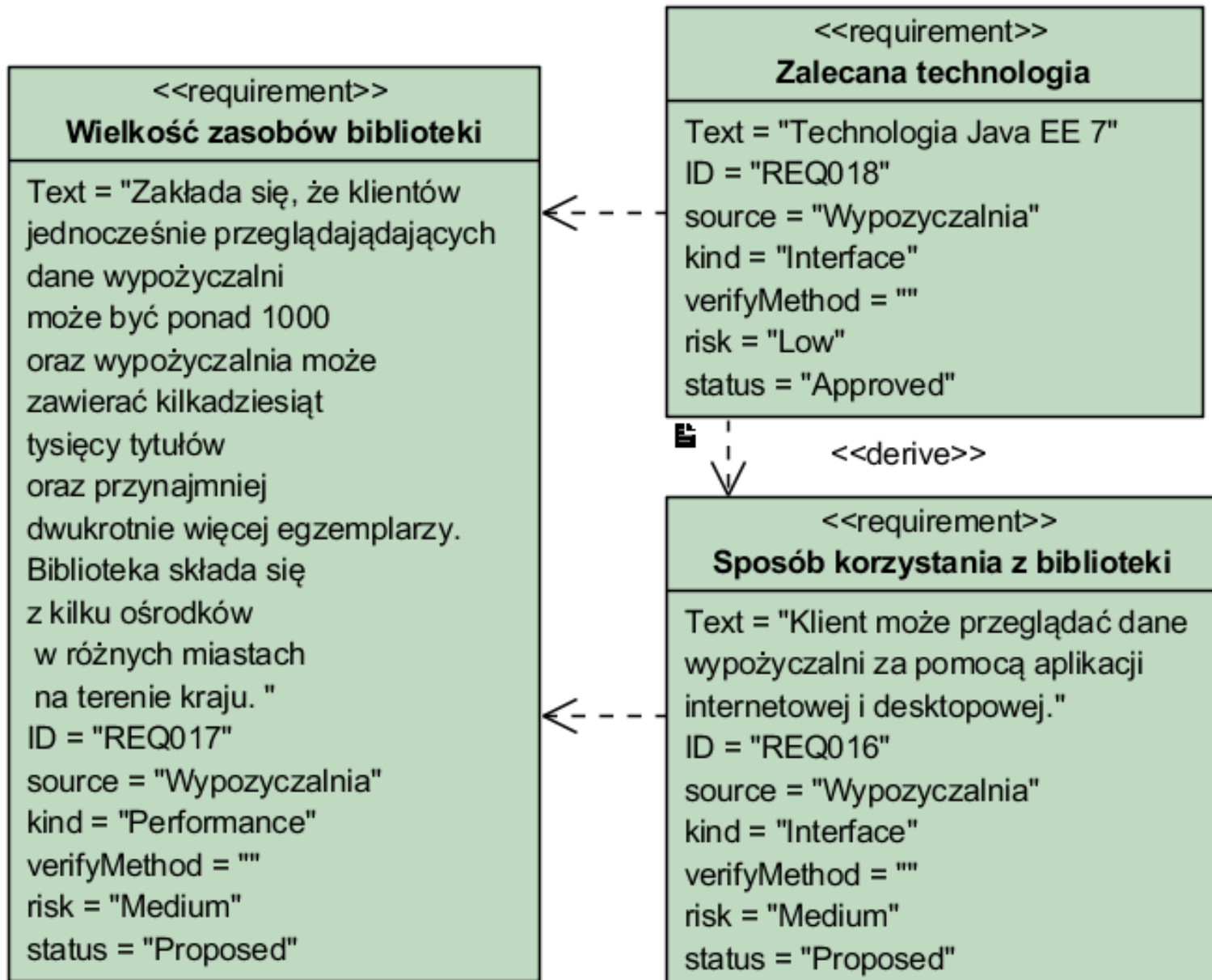


Diagram wymagań niefunkcjonalnych (cd)



III. Model analizy aplikacji oparty na diagramie przypadków użycia

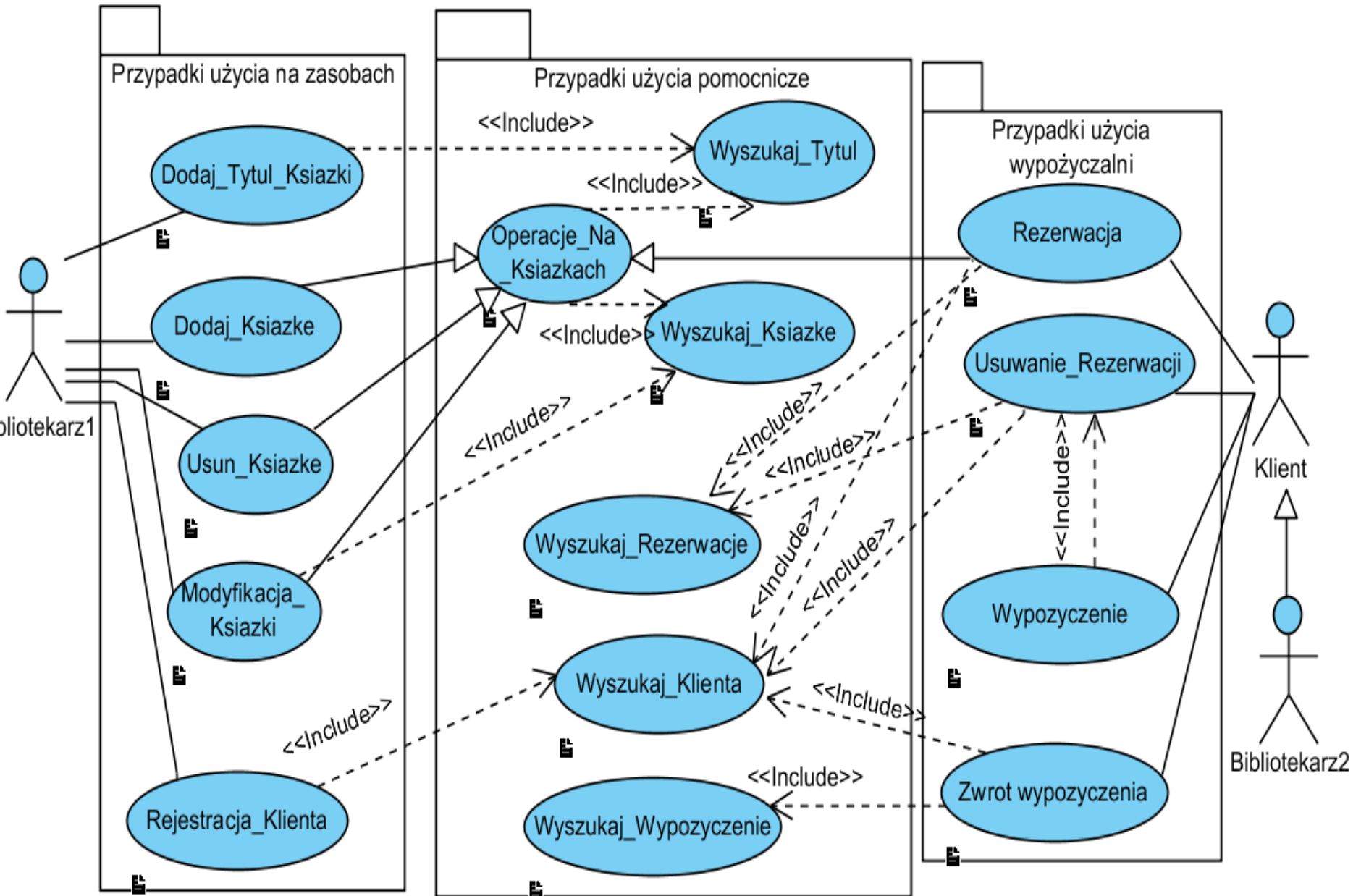
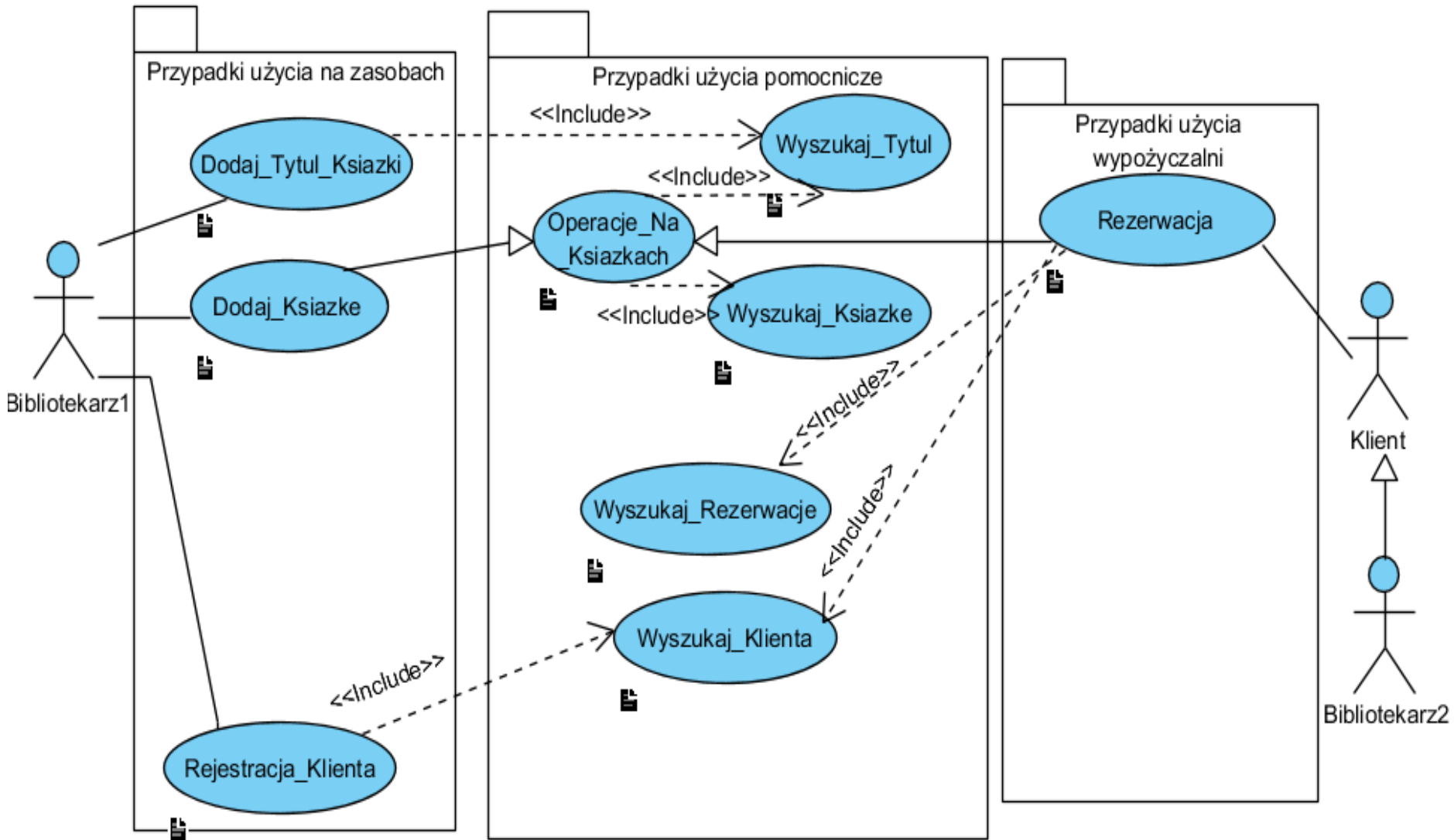


Diagram przypadków użycia – wybrany fragment



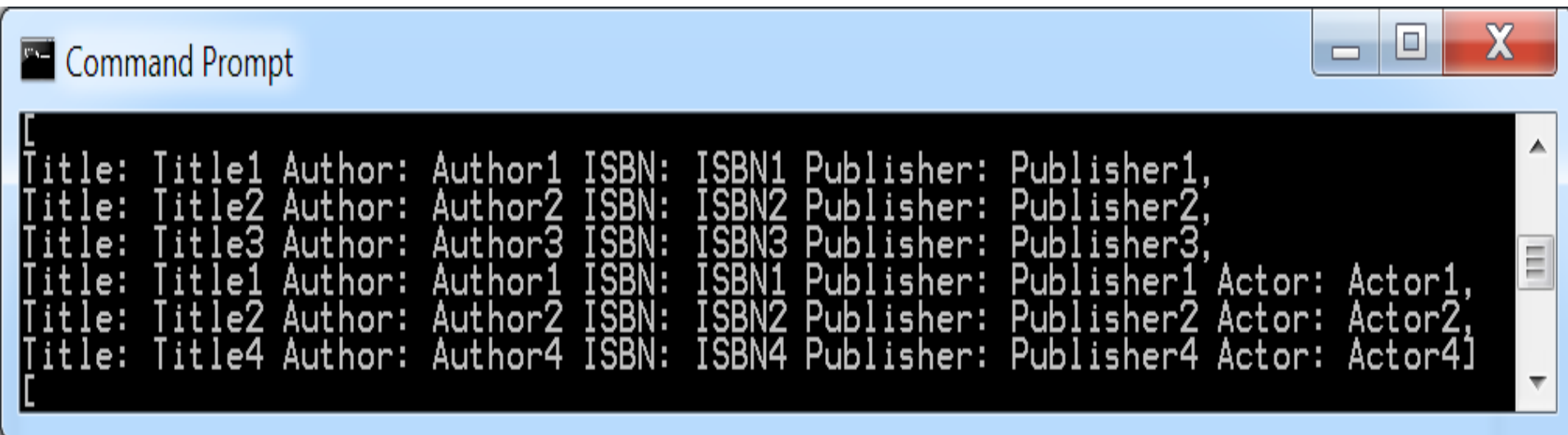
**IV. Model projektowy i implementacja warstwy
biznesowej warstwy biznesowej oparty na diagramie
klas i diagramie sekwencji tworzony metodą
iteracyjno-rozwojową, sterowany realizacją
przypadków użycia**

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.



```
[
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.

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]
```

Iteracja 3

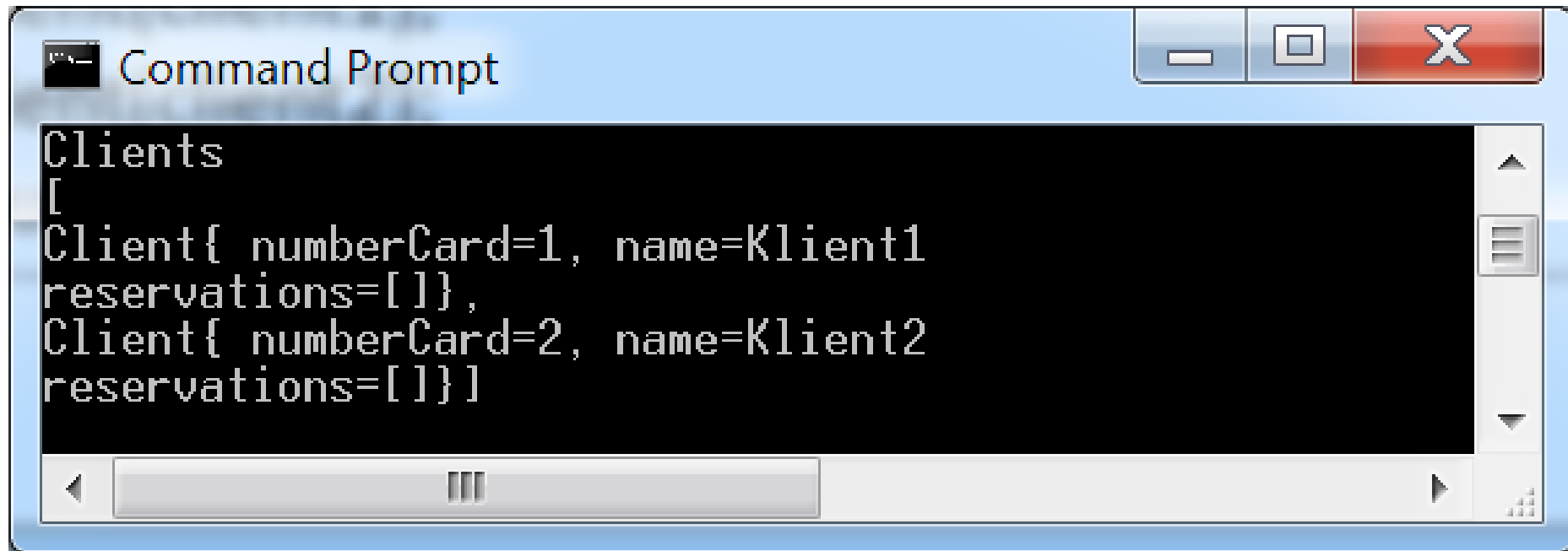
Projekt przypadku użycia

„ **Rejestracja_Klienta** ”

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

//dodawanie klientów

```
System.out.println("\nClients");  
String[] client1 = {"1", "Klient1", "1"}, dclient1 = {"0", "1"};  
String[] client2 = {"1", "Klient2", "2"}, dclient2 = {"0", "2"}, dclient3 = {"0", "3"};  
ap.addClient(client1);  
ap.addClient(client1);  
ap.addClient(client2);  
System.out.println(ap.clients);
```



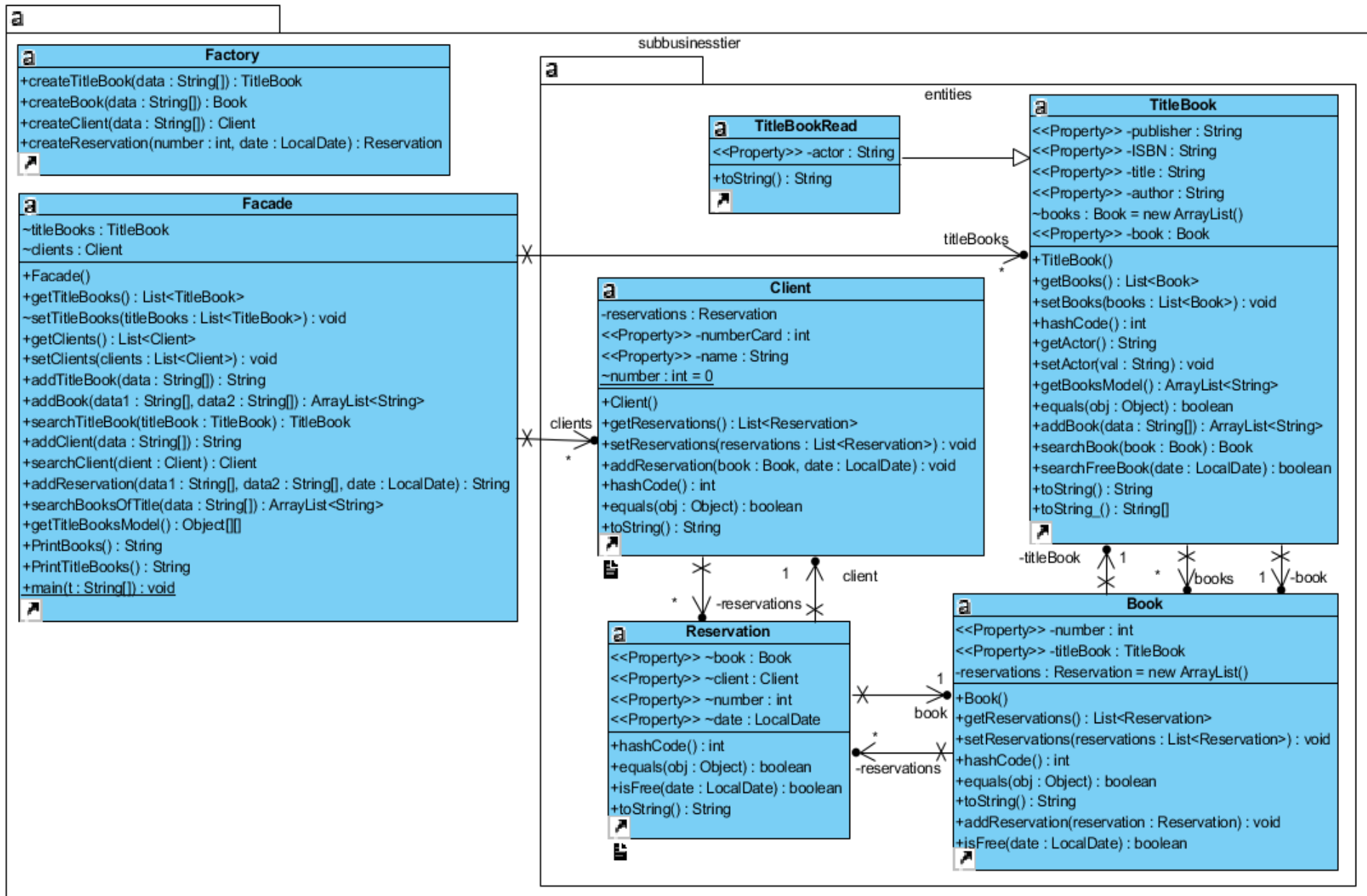
```
Command Prompt  
Clients  
[  
Client{ numberCard=1, name=Klient1  
reservations=[]},  
Client{ numberCard=2, name=Klient2  
reservations=[]}]
```

Iteracja 4

Projekt przypadku użycia „**Rezerwacja**”

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

Rezultat – diagram klas uzyskany w procesie projektowania (przebieg pokazany w dodatku do wykładu 5)



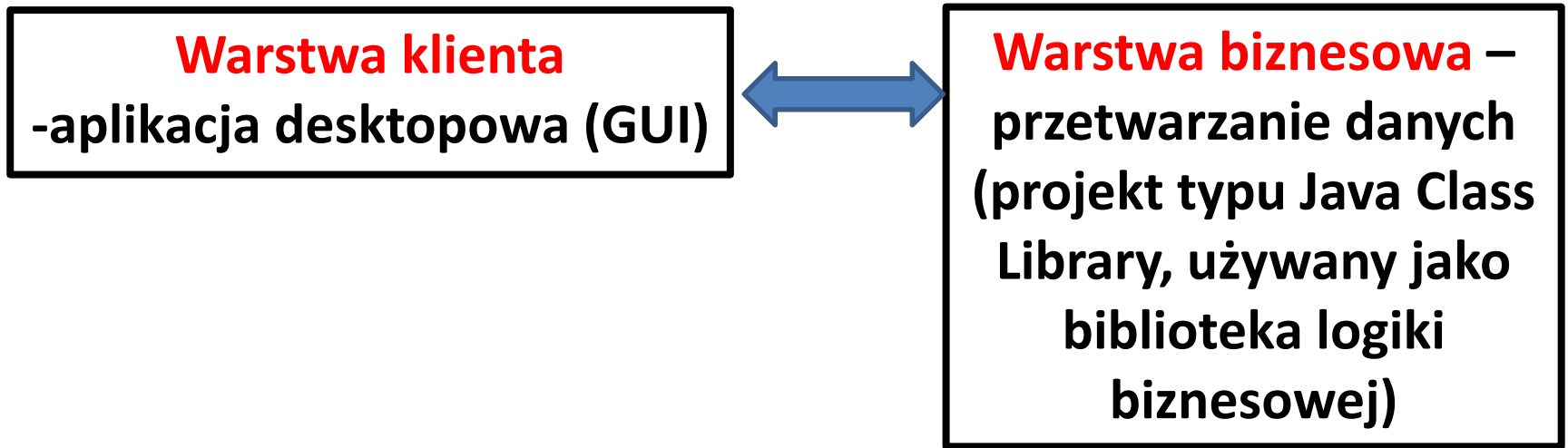
```
Command Prompt

Clients
[
Client{ numberCard=1, name=Klient1
reservations=[]},
Client{ numberCard=2, name=Klient2
reservations=[]}]

Reservations
reserved
no free book
reserved
no such a client
reserved
no free book

Clients
[
Client{ numberCard=1, name=Klient1
reservations=[Reservation{ number=0, date=2018-01-20}, Reservation{ number=2, date=2018-01-20}]},
Client{ numberCard=2, name=Klient2
reservations=[Reservation{ number=1, date=2018-01-20}]}]
```


Wykonanie aplikacji dwuwarstwowej



Projekt typu Java
Class Library
zawierający kod
warstwy biznesowej
wykonany podczas
4 iteracji

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

Library1_client1_SE - NetBeans IDE 8.2

Search (Ctrl+I)

Projects Files Services

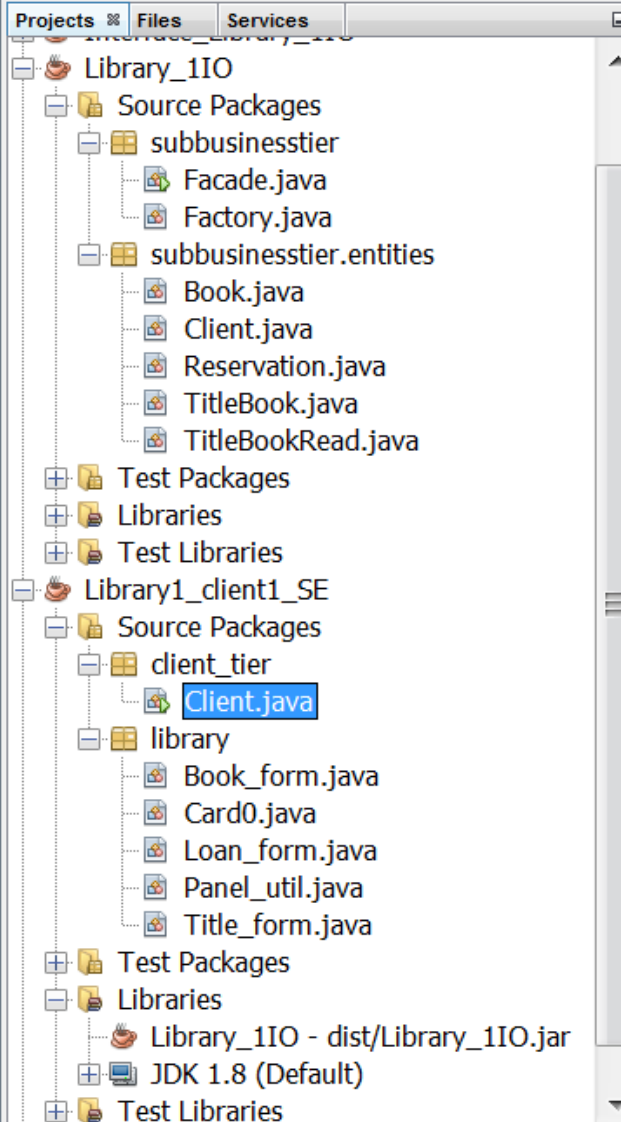
- Library_1IO
 - Source Packages
 - subbusinessstier
 - Facade.java
 - Factory.java
 - subbusinessstier.entities
 - Book.java
 - Client.java
 - Reservation.java
 - TitleBook.java
 - TitleBookRead.java
 - Test Packages
 - Libraries
 - Test Libraries
- Library1_client1_SE
 - Source Packages
 - client_tier
 - 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 (Default)
 - Test Libraries

Output

java DB Database Process

```
run:  
BUILD SUCCESSFUL (t
```

Projekt
zawierający
warstwę
biznesową jest
biblioteką dla
projektu **warstwy**
klienta



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

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

client_tier.Client > main >

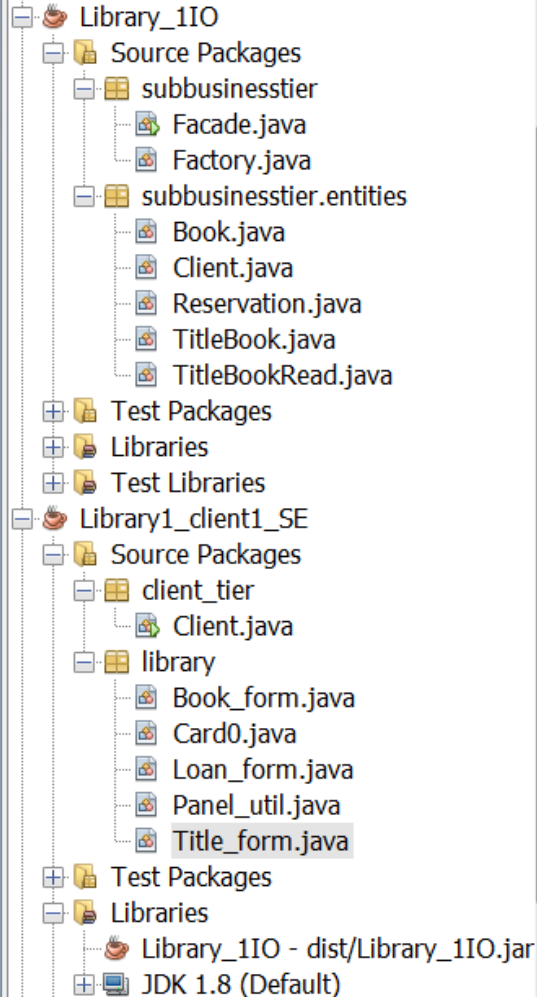
Output

>> Java DB Database Process Library1_client1_SE (run)



17:47

INS



```
46
47 public void init() {
48     add_title.addActionListener(this);
49 }
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(title) == null) {
62         return null;
63     }
64     if (content_validate(ISBN) == null) {
```

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

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

```
Object[][] titles = Client.getFacade().getTitleBooksModel();
```

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);
```

```
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);
    }
}
```

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

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

MenuDemo

A Menu Another Menu

Title
Tytul1

Author
Author1

ISBN
1234567

Publisher
Publisher1

Actor
Actor1

Add title

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Publisher1	1234567	Tytul1	Author1	Actor1

Number of a book

Add book

Books

MenuDemo

A Menu Another Menu

Title
Tytul1

Author
Author1

ISBN
1234567

Publisher
Publisher1

Actor

Add title

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Publisher1	1234567	Tytul1	Author1	Actor1
Publisher1	1234567	Tytul1	Author1	

Number of a book

Add book

Books

Wprowadzanie danych książek papierowych i nagranych

The application window 'MenuDemo' contains a menu with 'A Menu' and 'Another Menu'. The main area features a table with columns: Publisher, ISBN, Title, Author, and Actor. Below the table is a text input field labeled 'Number of a book' and an 'Add book' button. At the bottom, a 'Books' section displays a list of added books.

Initial State (Screenshot 1):

Publisher	ISBN	Title	Author	Actor
Publisher1	1234567	Tytul1	Author1	Actor1
Publisher1	1234567	Tytul1	Author1	Actor1

Number of a book: 1

Books: (empty)

After Clicking 'Add book' (Screenshot 2):

Publisher	ISBN	Title	Author	Actor
Publisher1	1234567	Tytul1	Author1	Actor1
Publisher1	1234567	Tytul1	Author1	Actor1

Number of a book: 1

Books: Title: Tytul1 Author: Author1 ISBN: 1234567 Publisher: Publisher1 Actor: Actor1 Number: 1

Final State (Screenshot 3):

Publisher	ISBN	Title	Author	Actor
Publisher1	1234567	Tytul1	Author1	Actor1
Publisher1	1234567	Tytul1	Author1	Actor1
Publisher1	1234567	Tytul1	Author1	Actor1

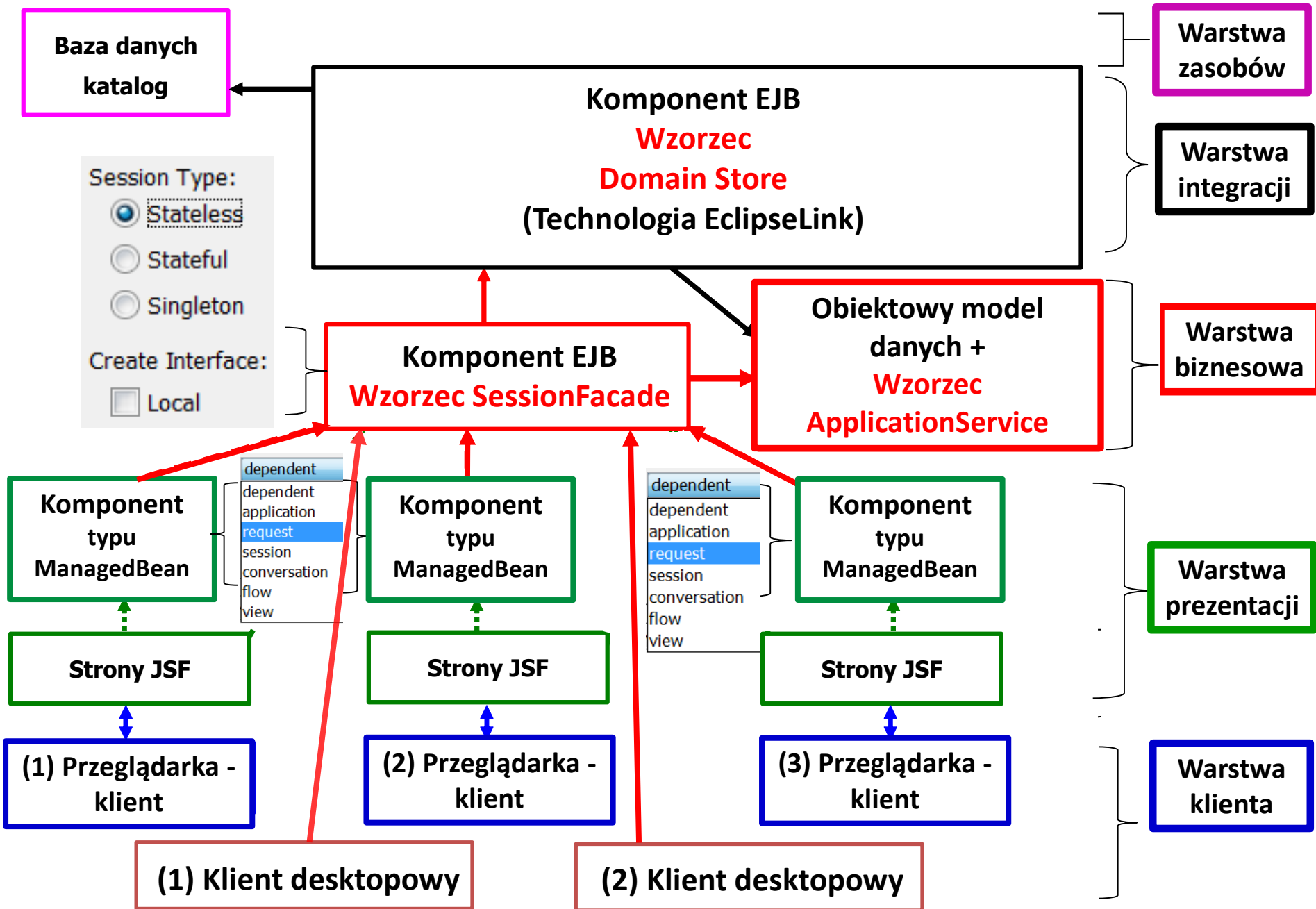
Number of a book: 2

Books: Title: Tytul1 Author: Author1 ISBN: 1234567 Publisher: Publisher1 Actor: Actor1 Number: 1
Title: Tytul1 Author: Author1 ISBN: 1234567 Publisher: Publisher1 Actor: Actor1 Number: 1
Title: Tytul1 Author: Author1 ISBN: 1234567 Publisher: Publisher1 Actor: Actor1 Number: 2

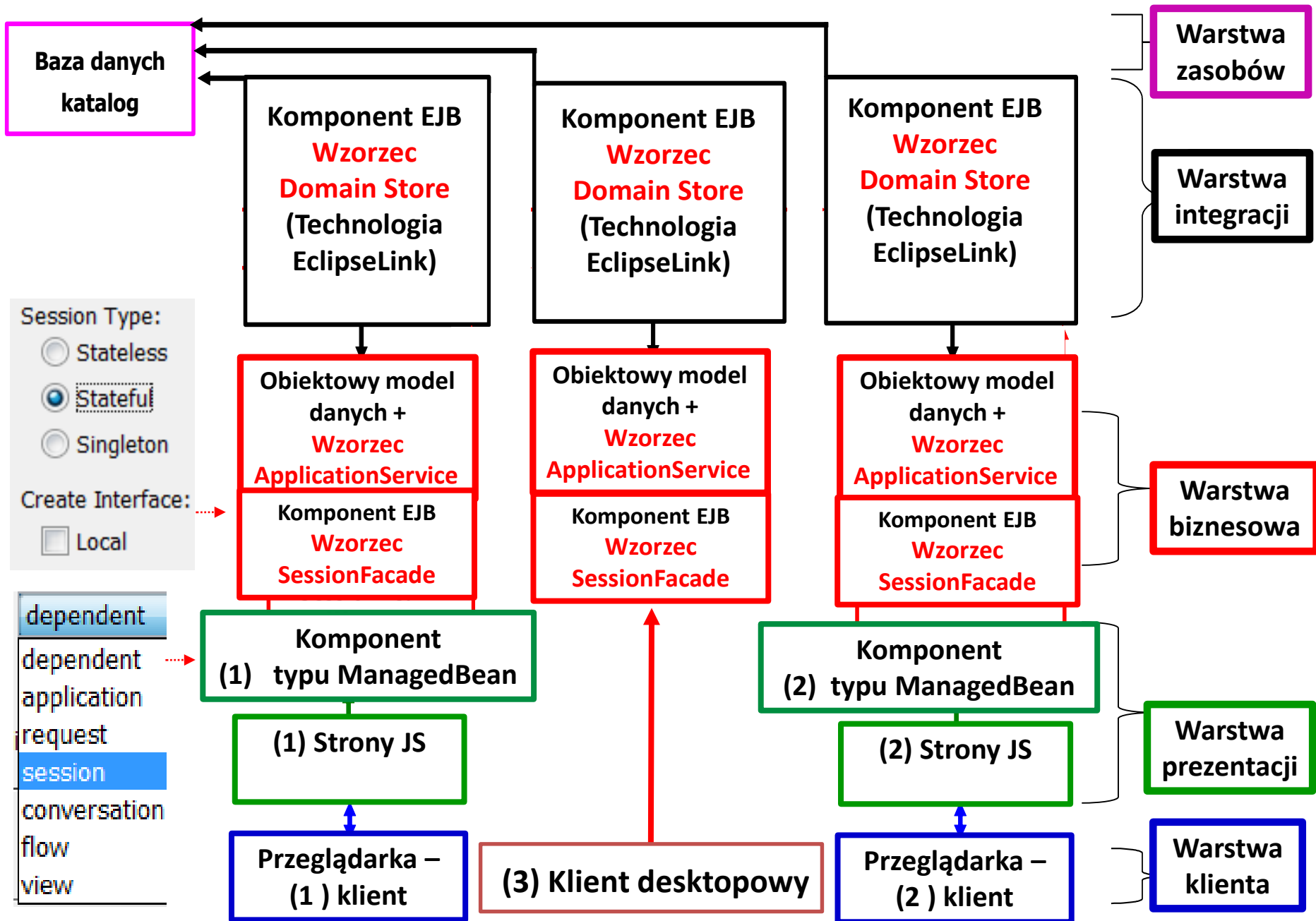
Zagadnienia

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

Architektura aplikacji pięciowarstwowej -Java EE 7.0 JavaServer Faces

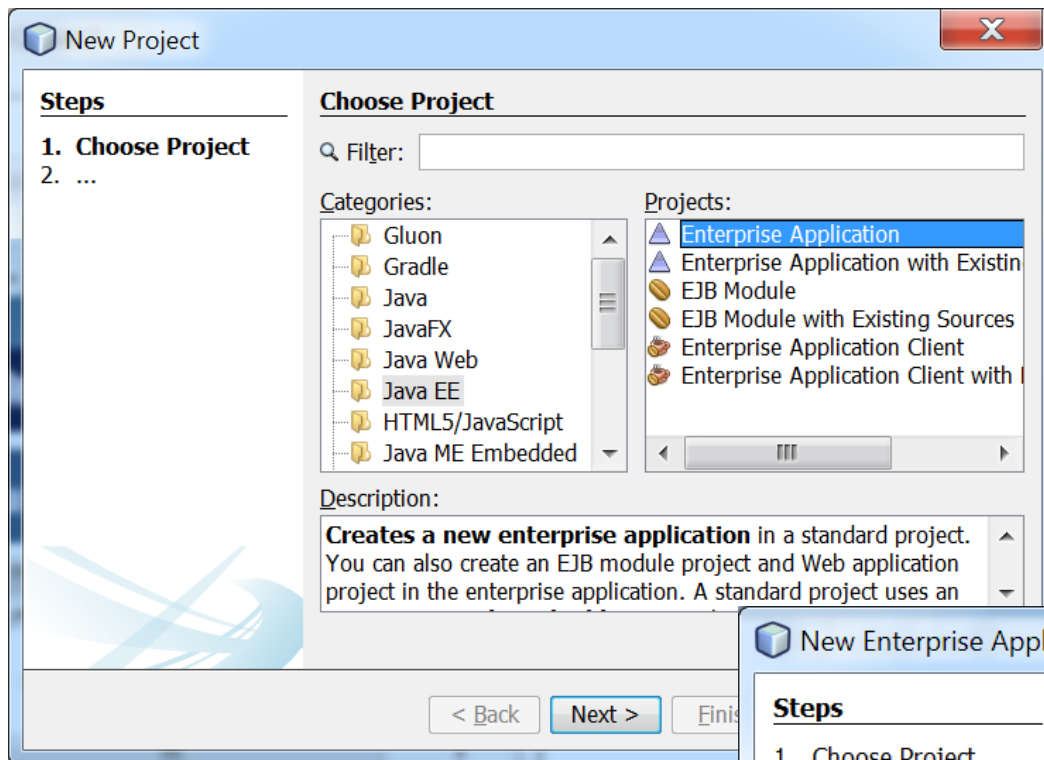


Architektura aplikacji pięciowarstwowej – Java EE 7.0 JavaServer Faces



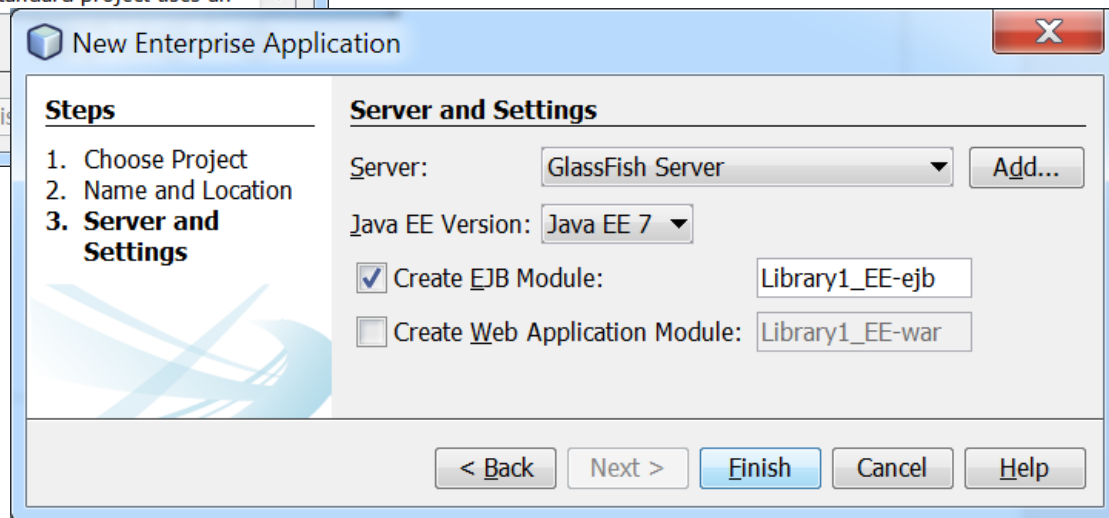
Wykonanie aplikacji typu Java EE z modułem EJB.

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

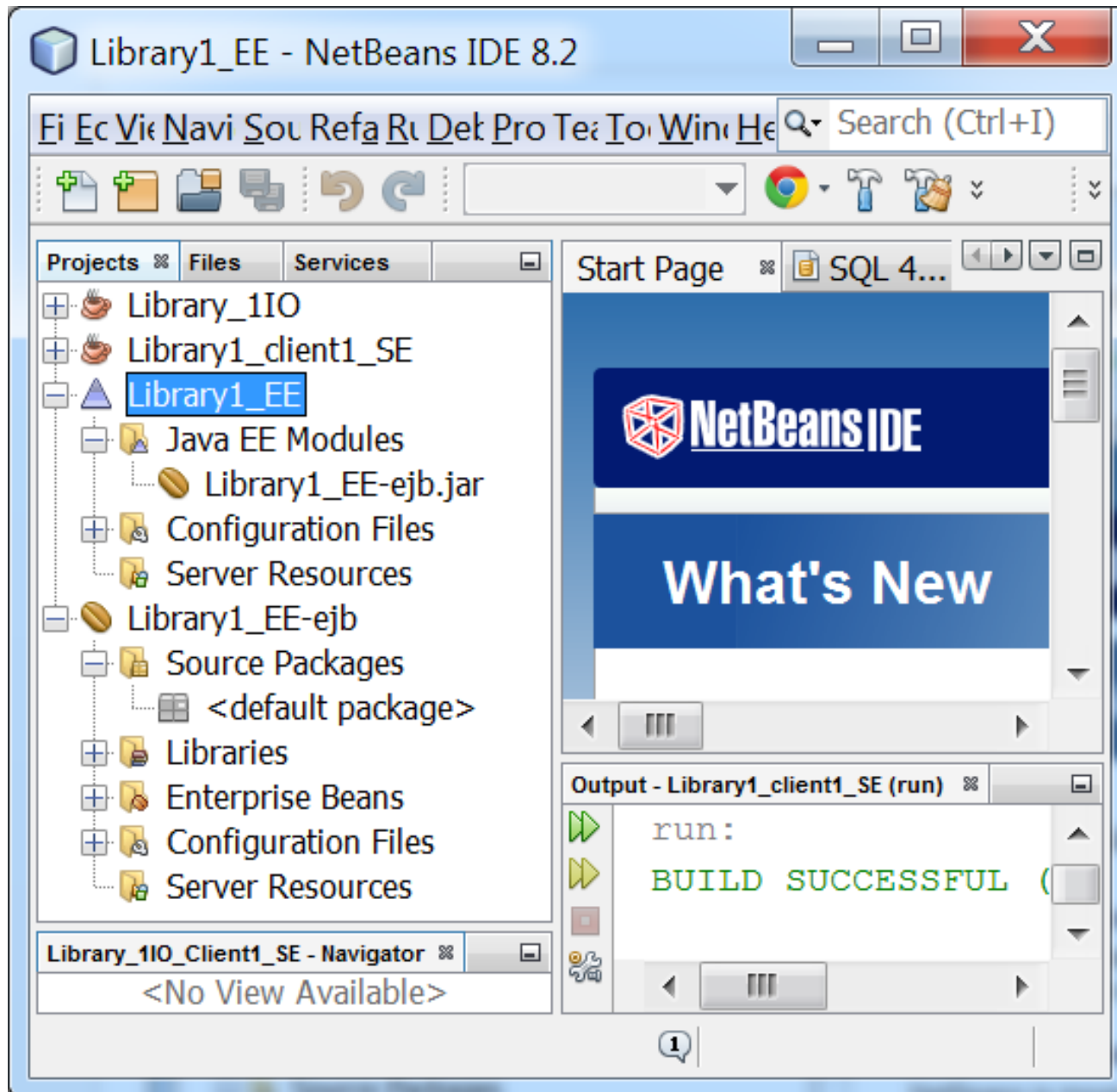


1. Tworzenie aplikacji typu Enterprise Application

2. Dodanie modułu EJB w aplikacji Enterprise Application



3. Projekt typu Enterprise Application



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

1. Choose Project

2. ...

Filter:

Categories:

- Glueon
- Gradle
- Java
- JavaFX
- Java Web

Projects:

- Java Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

Description:

Creates a new Java SE library in a standard IDE project. A Java SE library does not contain a main class. Standard projects

Name:

Location:

Folder:

Dedicated Folder for Storing Libraries

Location:

Different users and projects can share the same compilation libraries (see Help for details).

< Back Next > Finish Cancel Help

1. Choose File Type

2. ...

Project:

Filter:

Categories:

- JMX
- Bean Validation
- Enterprise JavaBean
- Contexts and Depen
- Java

File Types:

- Session Bean
- Timer Session Bean
- Message-Driven Bean
- Service Locator
- Caching Service Locator
- Session Beans For Entity Cl
- Standard Deployment Desc

Description:

Creates an empty Session Enterprise JavaBean (EJB) component. A session bean is typically used to encapsulate business logic and enterprise resources. This template creates the Java classes for...

< Back Next > Finish Cancel

New Session Bean

Steps

1. Choose File Type
2. Name and Location

Name and Location

EJB Name:

Project:

Location:

Package:

Session Type:

- Stateless
- Stateful
- Singleton

Create Interface:

- Local
- Remote in project:

< Back Next > Finish Cancel Help

5. Wygenerowany: zdalny komponent typu Session Bean i jego interfejs

The screenshot shows the NetBeans IDE 8.2 interface for the 'Library_1IO_interface' project. The 'Projects' window on the left displays the project structure, including 'Library_1IO', 'Library_1IO_interface', and 'Source Packages'. The 'EJBFacadeRemote.java' file is selected in the 'Files' window. The main editor displays the following code:

```
1 package busnesstier;
2
3
4 import javax.ejb.Remote;
5
6
7 @Remote
8 public interface EJBFacadeRemote {
9 }
10
```

The screenshot shows the NetBeans IDE 8.2 interface for the 'Library1_EE-ejb' project. The 'Projects' window on the left displays the project structure, including 'Library1_client1_SE', 'Library1_EE', 'Library1_EE-ejb', and 'Source Packages'. The 'EJBFacade.java' file is selected in the 'Files' window. The main editor displays the following code:

```
1 package busnesstier;
2
3 import javax.ejb.Stateless;
4
5 @Stateless
6 public class EJBFacade implements EJBFacadeRemote {
7
8
9 }
10
```

The 'Output' window at the bottom shows the following message:

```
run:
BUILD SUCCESSFUL (total time: 8 seconds)
```

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

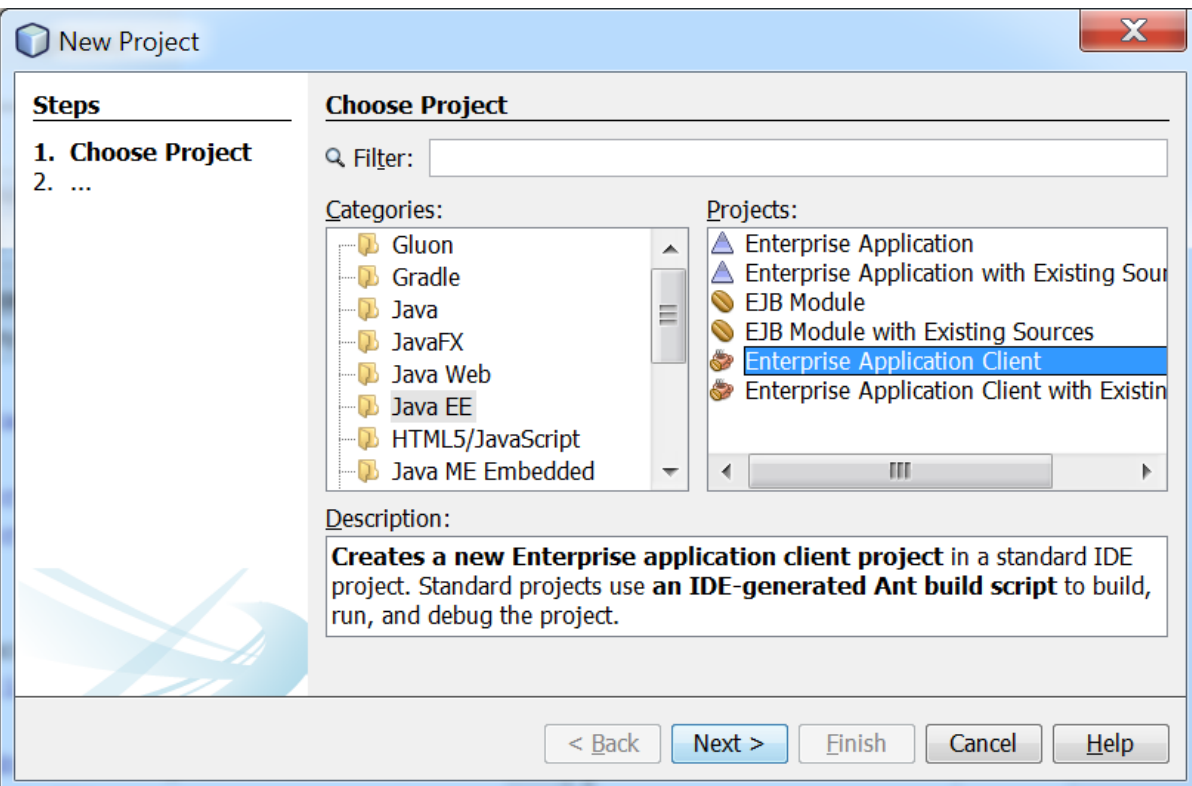
The screenshot shows an IDE window with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'Library1_EE-ejb' with a package 'businesstier' containing 'EJBFacadeRemote.java'. The code editor displays the following Java code:

```
1 package businesstier;
2 import java.util.ArrayList;
3 import javax.ejb.Remote;
4
5
6 @Remote
7 public interface EJBFacadeRemote {
8     public String addTitleBook(String data[]);
9     public ArrayList<String> addBook(String data1[], String data2[]);
10    public ArrayList<String> searchBooksOfTitle(String data[]);
11    public Object[][] getTitleBooksModel();
12 }
13
```


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

```
1 package businesstier;
2 import java.util.ArrayList;
3 import javax.ejb.Stateless;
4 import subbusinesstier.Facade;
5
6 @Stateless
7 public class EJBFacade implements EJBFacadeRemote {
8     Facade facade = new Facade();
9
10    @Override
11    public String addTitleBook(String data[])
12    { return facade.addTitleBook(data); }
13
14    @Override
15    public ArrayList<String> addBook(String data[], String data2[])
16    { return facade.addBook(data1, data2); }
17
18    @Override
19    public ArrayList<String> searchBooksOfTitle(String data[])
20    { return facade.searchBooksOfTitle(data); }
21
22    @Override
23    public Object[][] getTitleBooksModel()
24    { return facade.getTitleBooksModel(); }
25
26 }
```

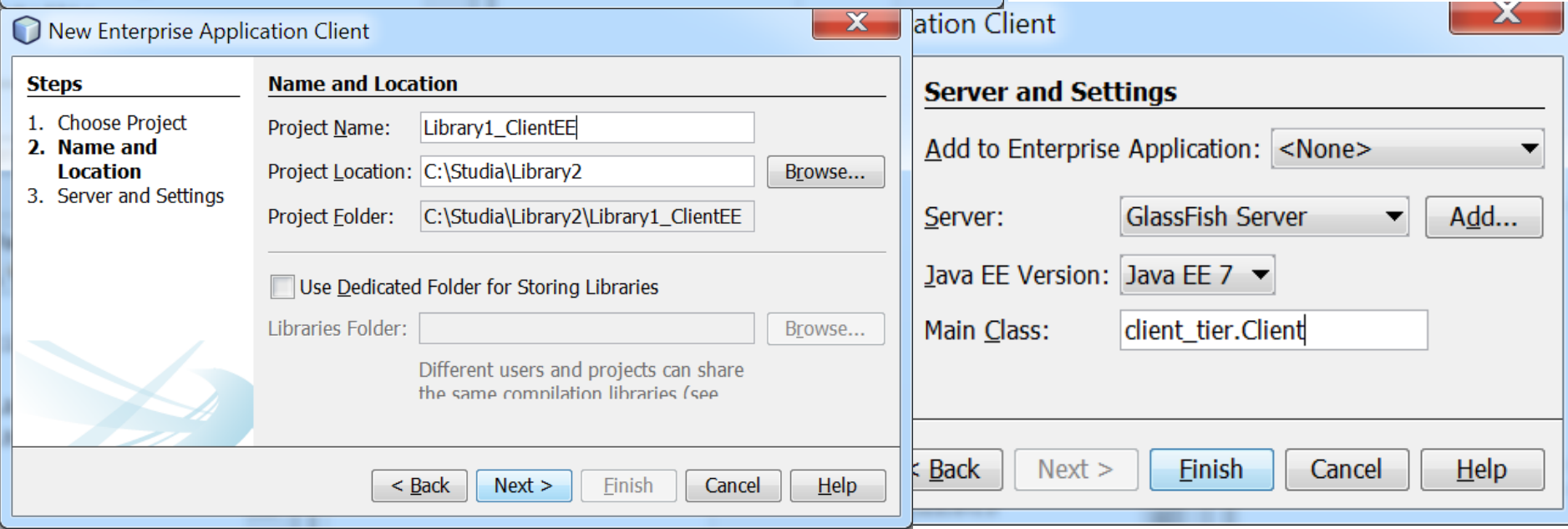
Obiekt typu *Facade* pełni rolę wzorca *ApplicationService*



8. Wykonanie projektu typu **Enterprise Application Client**.

Kod tego projektu będzie wykonany w oparciu o pakiet **library** z projektu warstwy klienta (aplikacja dwuwarstwowa).

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 typie **Enterprise Application Client** (w kodzie klasy **Client**)

The screenshot shows the NetBeans IDE 8.2 interface. The main window displays the source code of the `Client.java` file in the `client_tier` package. A context menu is open over the `Client` class, with the `Call Enterprise Bean...` option selected. A secondary dialog box, `Call Enterprise Bean`, is open, showing the selection of the `EJBFacade` bean from the `Library1_EE-ejb` project. The `Reference Name` is set to `EJBFacade`, and the `Referenced Interface` is set to `Remote`.

The IDE interface includes the following elements:

- Project Explorer:** Shows the project structure with `Library1_ClientEE` and its sub-packages, including `client_tier` and `library`.
- Source Editor:** Displays the code for `Client.java`, showing the `package client_tier;` declaration and the `public class Client` definition.
- Context Menu:** Lists actions such as `Navigate`, `Show Javadoc`, `Find Usages`, `Call Hierarchy`, `Insert Code...`, `Fix Imports`, `Refactor`, `Format`, `Run File`, `Debug File`, `Test File`, `Debug Test File`, `Run Focused Test Method`, and `Debug Focused Test Method`.
- Call Enterprise Bean Dialog:** A dialog box for selecting an enterprise bean. It shows the `Library1_EE-ejb` project containing the `EJBFacade` bean. The `Reference Name` is `EJBFacade`, and the `Referenced Interface` is `Remote`.

11. Wykonana aplikacja reprezentująca warstwę klienta EE

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

```
4 import business-tier.EJBFacadeRemote;
5 import javax.ejb.EJB;
6 import library.Panel_util;
7
8 public class Client {
9
10     @EJB
11     private static EJBFacadeRemote facade;
12
13     public static EJBFacadeRemote getFacade() {
14         return facade;
15     }
16
17     public static void main(String[] args) {
18         java.awt.EventQueue.invokeLater(() -> {
19             Panel_util.createAndShowGUI();
20         });
21     }
22 }
```

Projects | Files | Services

- Libraries
- Test Libraries
- Library1_client1_SE
 - Source Packages
 - client_tier
 - Client.java
 - library
 - Test Packages
 - Libraries
 - Test Libraries
- Library1_ClientEE
 - Source Packages
 - client_tier
 - Client.java
 - library
 - Test Packages
 - Libraries
 - Test Libraries
 - Configuration Files

main - Navigator

Members <empty>

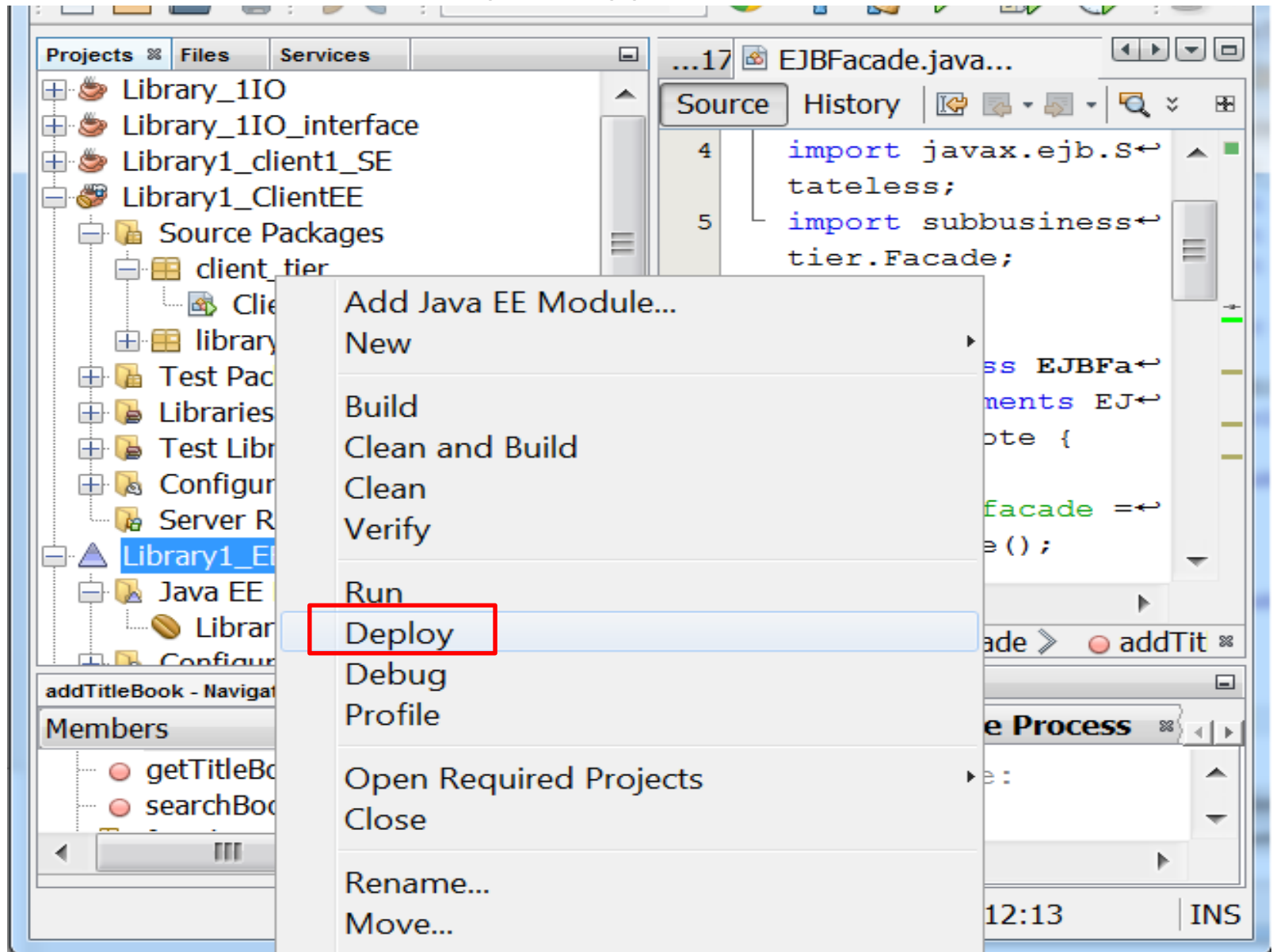
- main(String[] args)

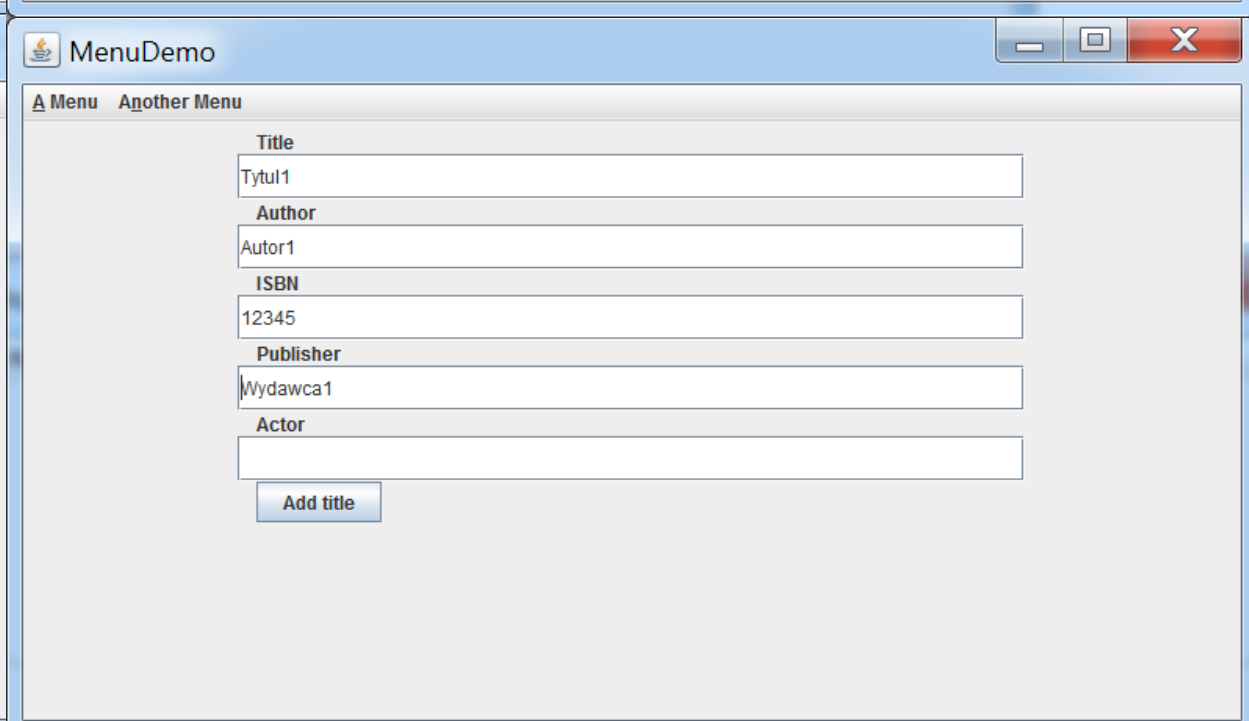
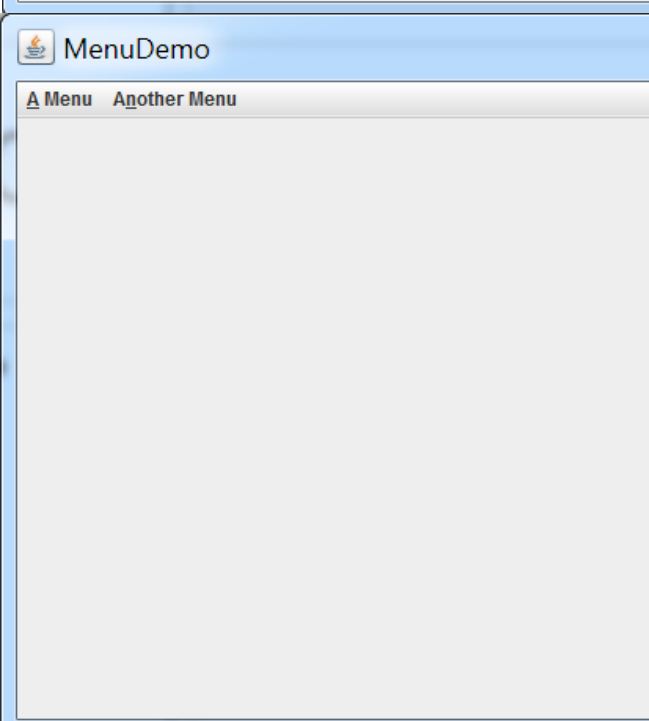
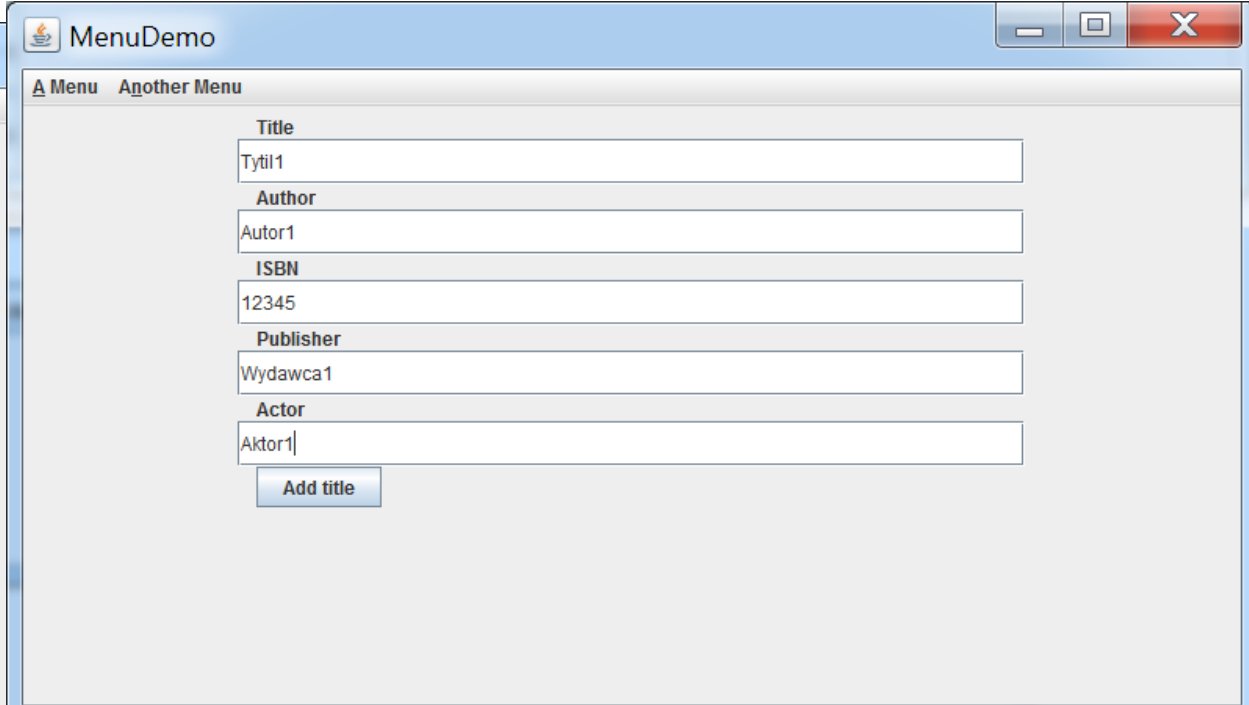
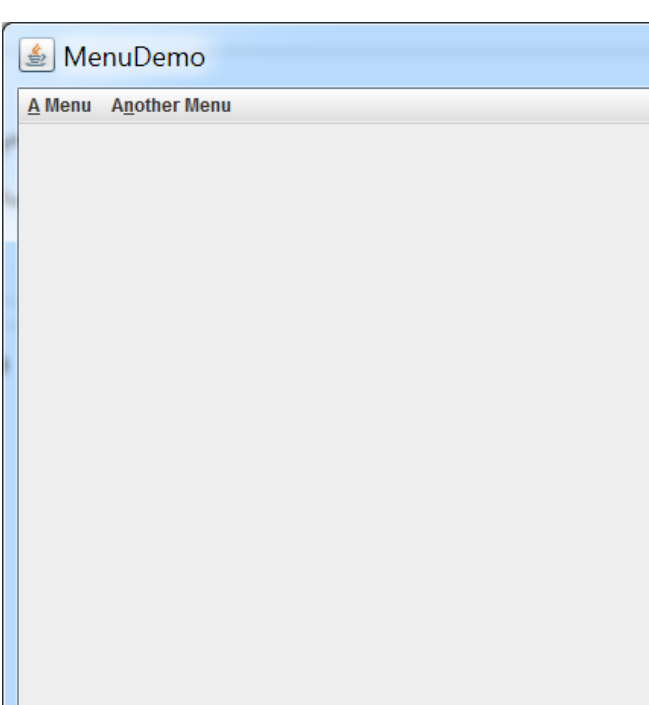
client_tier.Client > main >

Output - Library1_client1_SE (run)

```
>> run:
```

12. Uruchomienie desktopowej aplikacji wielowarstwowej typu EE: **Deploy** na aplikacji typu Enterprise Application, a potem **Run** na projekcie typu Enterprise Application Client







MenuDemo



A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	12345	Tytil1	Autor1	Aktor1
Wydawca1	12345	Tytil1	Autor1	

Number of a book

Add book

Clear selection

Books



MenuDemo



A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	12345	Tytil1	Autor1	Aktor1
Wydawca1	12345	Tytil1	Autor1	

Number of a book

Add book

Clear selection

Books

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	12345	Tytul1	Autor1	Aktor1
Wydawca1	12345	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

Title: Tytul1 Author: Autor1 ISBN: 12345 Publisher: Wydawca1 Actor: Aktor1 Number: 1

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	12345	Tytul1	Autor1	Aktor1
Wydawca1	12345	Tytul1	Autor1	

Number of a book

Add book

Clear selection

Books

MenuDemo

A Menu Another Menu

Publisher	ISBN	Title	Author	Actor
Wydawca1	12345	Tytul1	Autor1	Aktor1
Wydawca1	12345	Tytul1	Autor1	

Number of a book

Add book

Clear selection

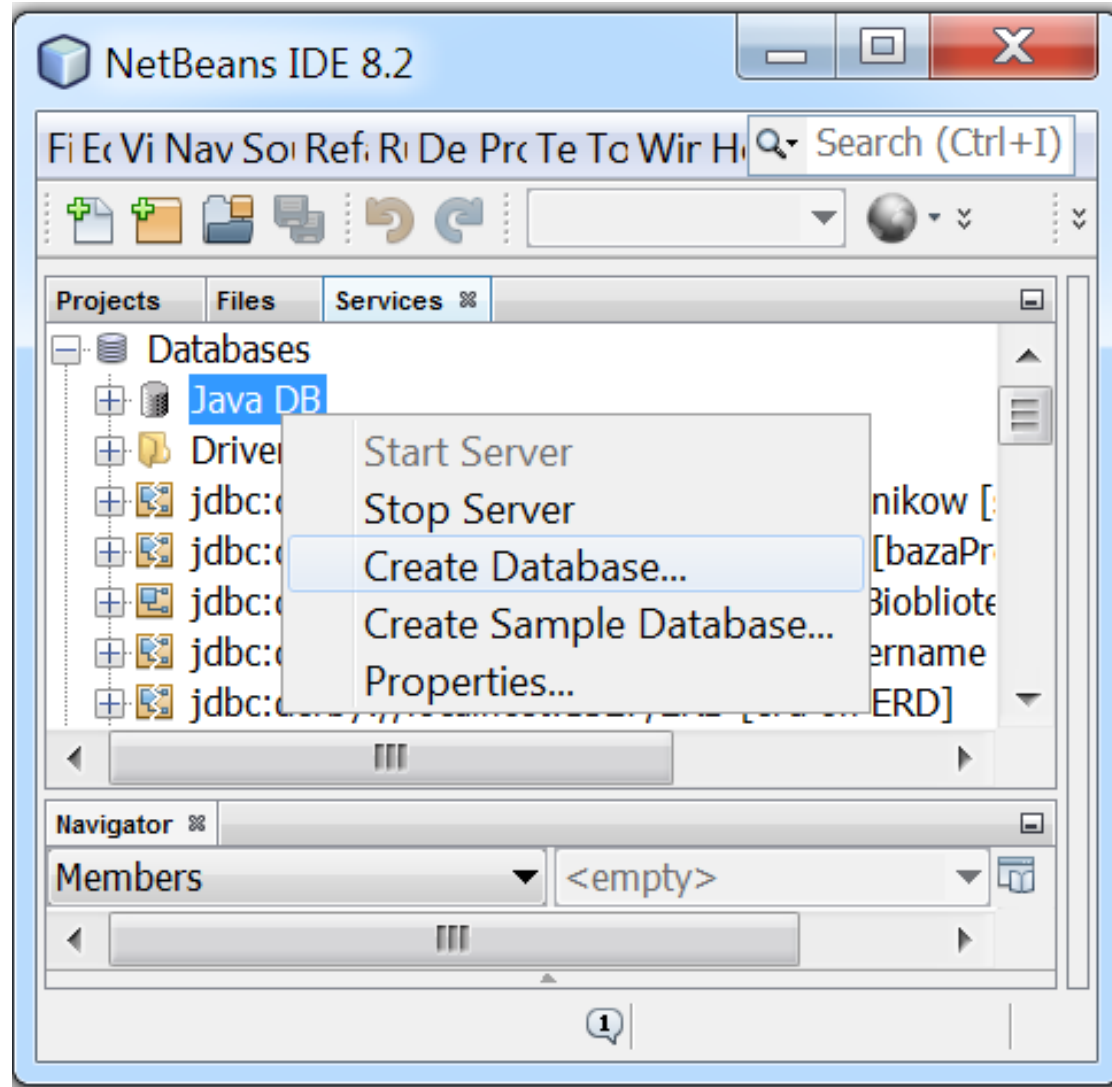
Books

Title: Tytul1 Author: Autor1 ISBN: 12345 Publisher: Wydawca1 Actor: Aktor1 Number: 1

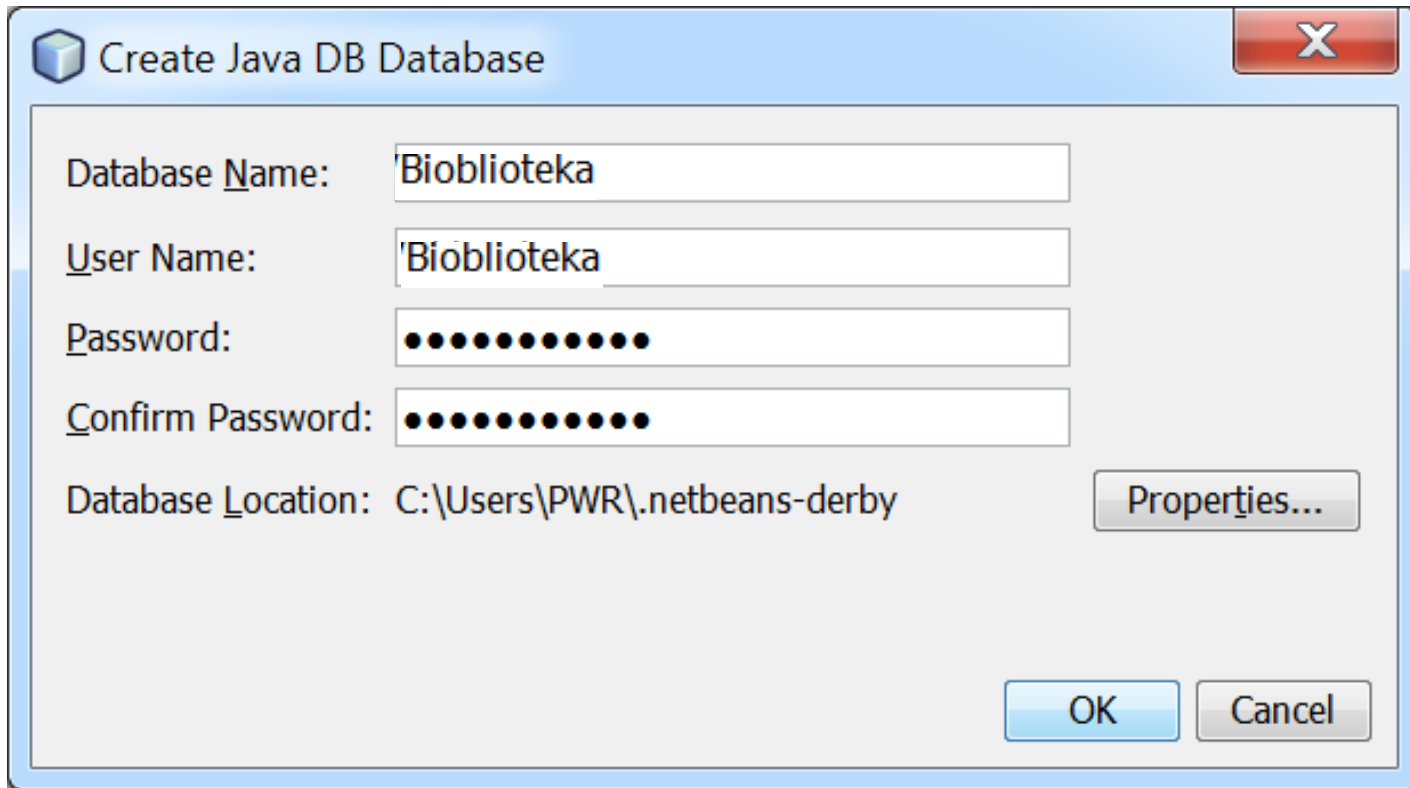
Zagadnienia

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

1. Zakładanie pustej bazy danych dla systemu baz danych Derby



2. Zakładanie pustej bazy danych **Bioblioteka** w systemie baz danych Derby



Create Java DB Database

Database Name: Bioblioteka

User Name: Bioblioteka

Password: ●●●●●●●●

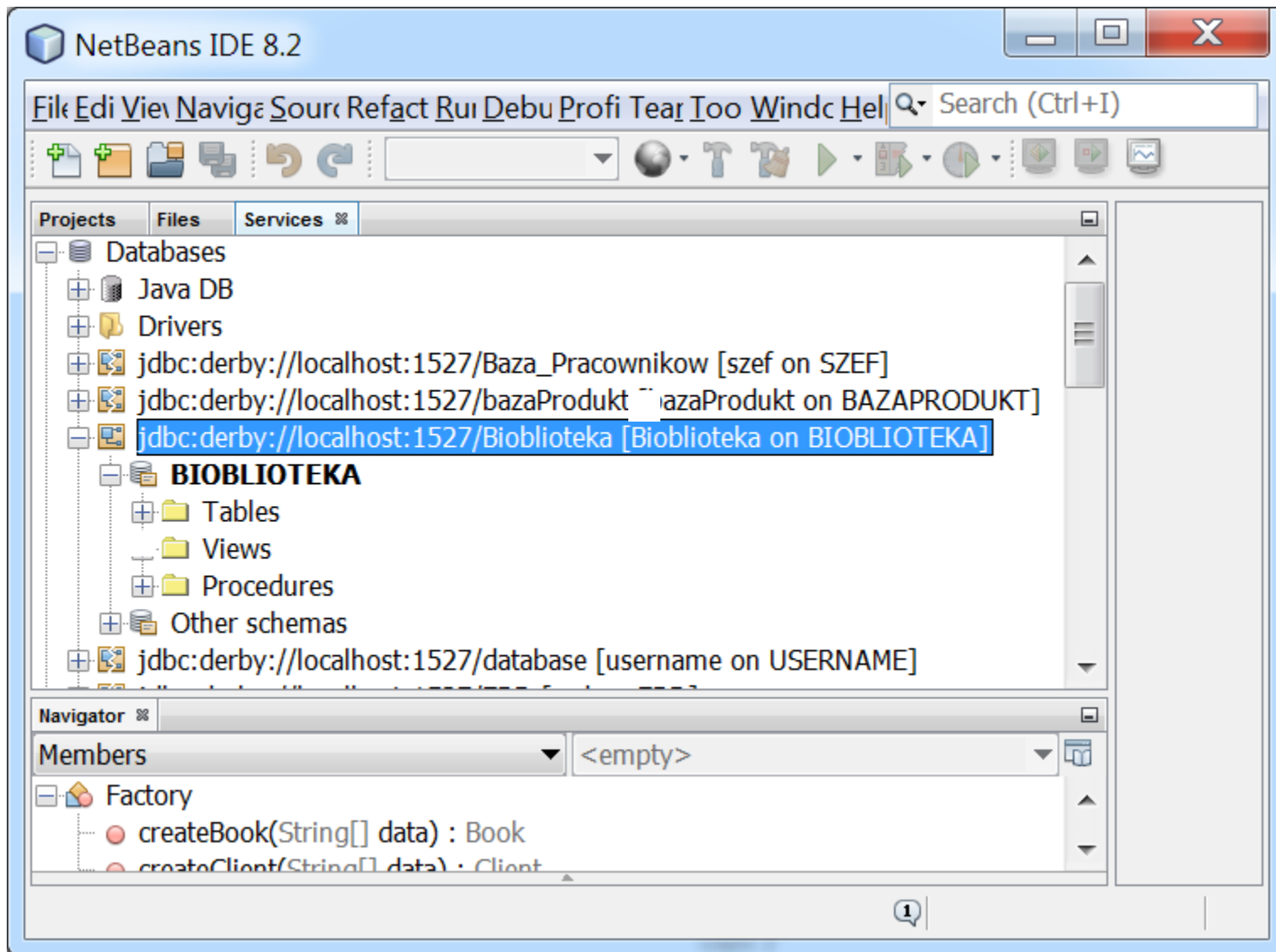
Confirm Password: ●●●●●●●●

Database Location: C:\Users\PWR\.netbeans-derby

Properties...

OK Cancel

3. Utworzona pusta baza danych



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. Przykład modelowania i projektowania **części warstwy biznesowej** z obiektami typu POJO. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
5. Przykłady architektury wielowarstwowej aplikacji internetowej typu EE. Wykonanie aplikacji typu EE. **Warstwa biznesowa:** komponenty typu EJB + obiekty POJO
6. **Warstwa zasobów** (EIS)- baza danych w systemie baz danych Derby
7. **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)
```

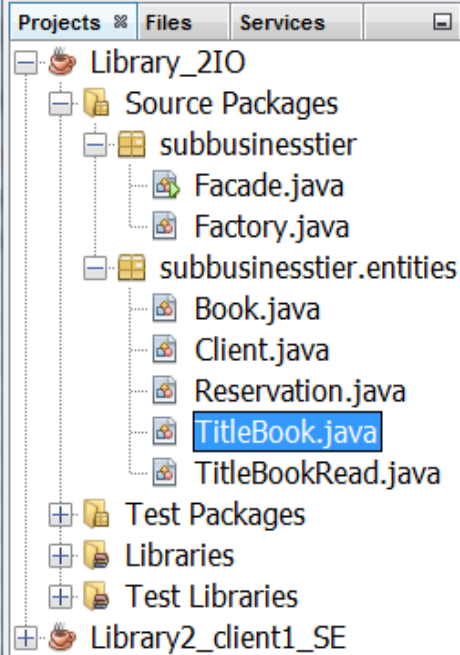
```
private Long id;
```

```
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 }
```



```
...va Book.java Factory.java Facade.java TitleBook.java
Source History
17
   @Entity
   public class TitleBook implements Serializable {
20     private static final long serialVersionUID=1L;
21     private String publisher;
22     private String ISBN;
23     private String title;
24     private String author;
25     @Id
26     @GeneratedValue(strategy = GenerationType.AUTO)
27     private Long id;
28     public Long getId() {...3 lines }
31     public void setId(Long id) {...3 lines }
34     @OneToMany(mappedBy = "titleBook", cascade=ALL)
35     List<Book> books;
36     public List<Book> getBooks() {...3 lines }
39     public void setBooks(List<Book> books) {...3 lin
42     public TitleBook() {...3 lines }
```


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. Przykład modelowania i projektowania **części warstwy biznesowej** z obiektami typu POJO. Wykonanie aplikacji dwuwarstwowej dla jednego użytkownika.
5. Przykłady architektury wielowarstwowej aplikacji internetowej typu EE. Wykonanie aplikacji typu EE. **Warstwa biznesowa**: komponenty typu EJB + obiekty POJO
6. **Warstwa zasobów (EIS)**- baza danych w systemie baz danych Derby
7. Utworzenie obiektowego modelu danych do utrwalania ORM
8. **Warstwa integracji. Zastosowanie wzorca projektowego typu *Domain Store* w technologii JPA (Java Persistence) na platformie Java EE**

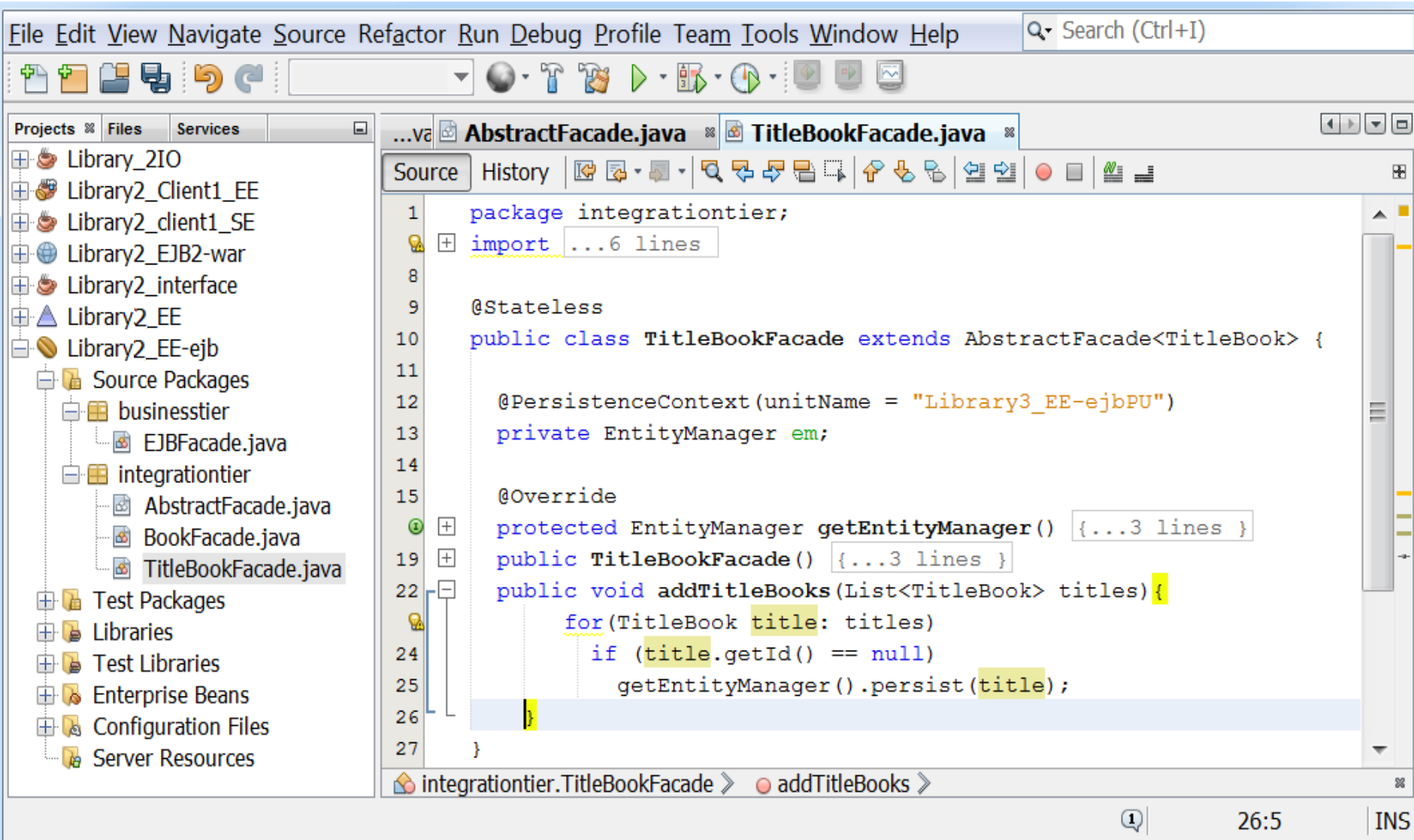
1. Klasa abstrakcyjna generyczna **AbstractFacade** z definicją uniwersalnych metod obsługujących transakcje JPA - parametr T może być zastąpiony każdą z klas typu Entity

The screenshot shows an IDE window with the following components:

- Menu Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. Search (Ctrl+I) is visible on the right.
- Toolbar:** Standard IDE icons for file operations, navigation, and execution.
- Project Explorer (Left):** Shows a project structure with packages like 'integrationtier' containing 'AbstractFacade.java', 'BookFacade.java', and 'TitleBookFacade.java'.
- Editor (Center):** Displays the source code of 'AbstractFacade.java'. The code is as follows:

```
1 package integrationtier;
2
3 import ...2 lines
4
5
6 public abstract class AbstractFacade<T> {
7     private Class<T> entityClass;
8     public AbstractFacade(Class<T> entityClass) {...3 lines }
9     protected abstract EntityManager getEntityManager();
10
11
12 public void create(T entity) {...3 lines }
13
14
15 public void edit(T entity) {...3 lines }
16
17
18 public void remove(T entity) {...3 lines }
19
20
21 public T find(Object id) {...3 lines }
22
23
24 public List<T> findAll() {...5 lines }
25
26
27
28
29 public List<T> findRange(int[] range) {...8 lines }
30
31
32
33
34
35
36
37 public int count() {...7 lines }
38
39
40
41
42
43
44 }
45
```
- Status Bar (Bottom):** Shows 'integrationtier.AbstractFacade' and zoom level '7:1'.

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

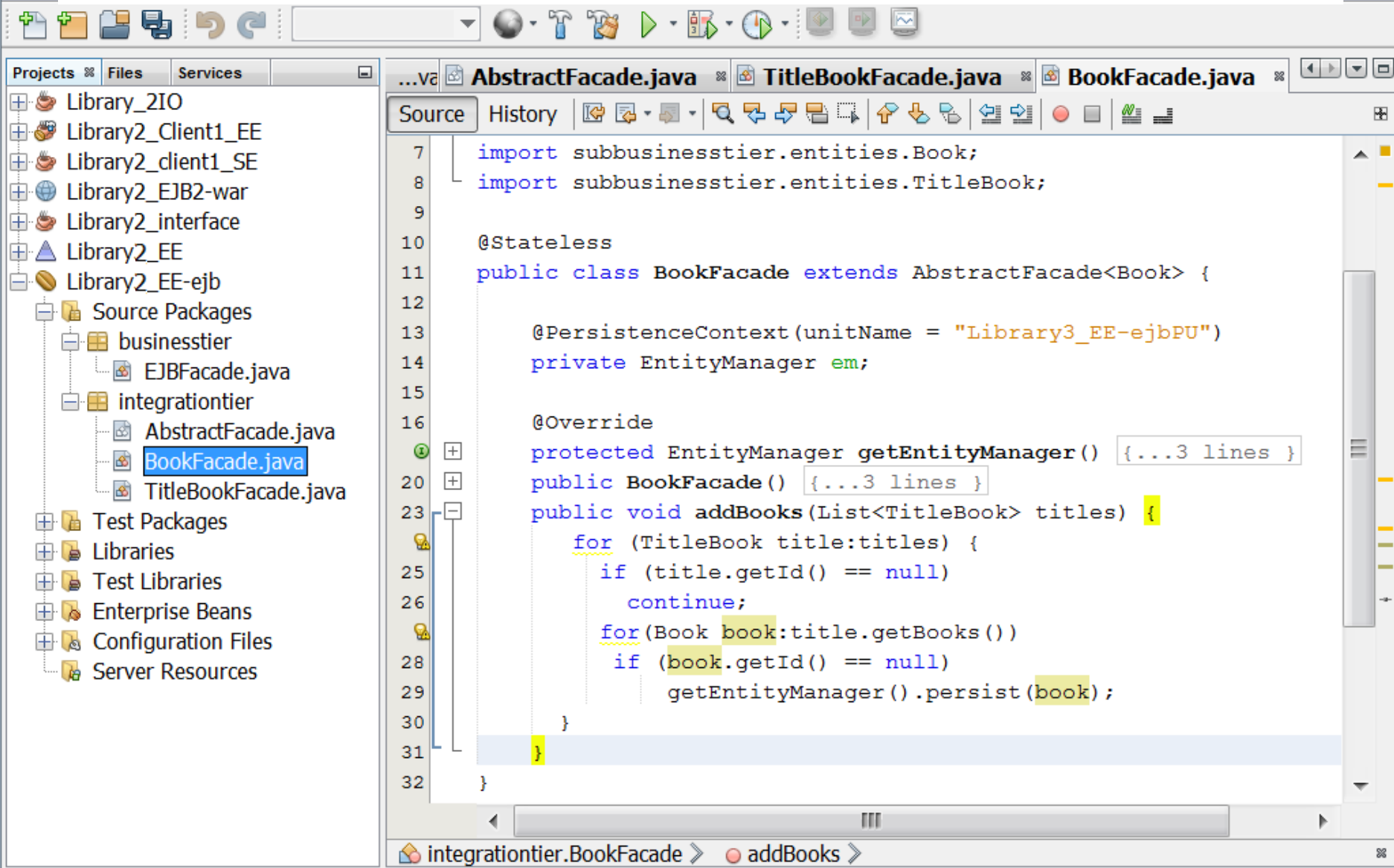


The screenshot displays an IDE window with the following components:

- Menu Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. A search bar on the right contains "Search (Ctrl+I)".
- Toolbar:** Contains icons for file operations (new, open, save, print), navigation (back, forward), and execution (run, debug).
- Project Explorer (Left):** Shows a project structure with packages like "Library2_2IO", "Library2_Client1_EE", "Library2_client1_SE", "Library2_EJB2-war", "Library2_interface", "Library2_EE", and "Library2_EE-ejb". Under "Source Packages", "integrationtier" contains "AbstractFacade.java", "BookFacade.java", and "TitleBookFacade.java".
- Editor (Center):** Displays the source code for "TitleBookFacade.java". The code is as follows:

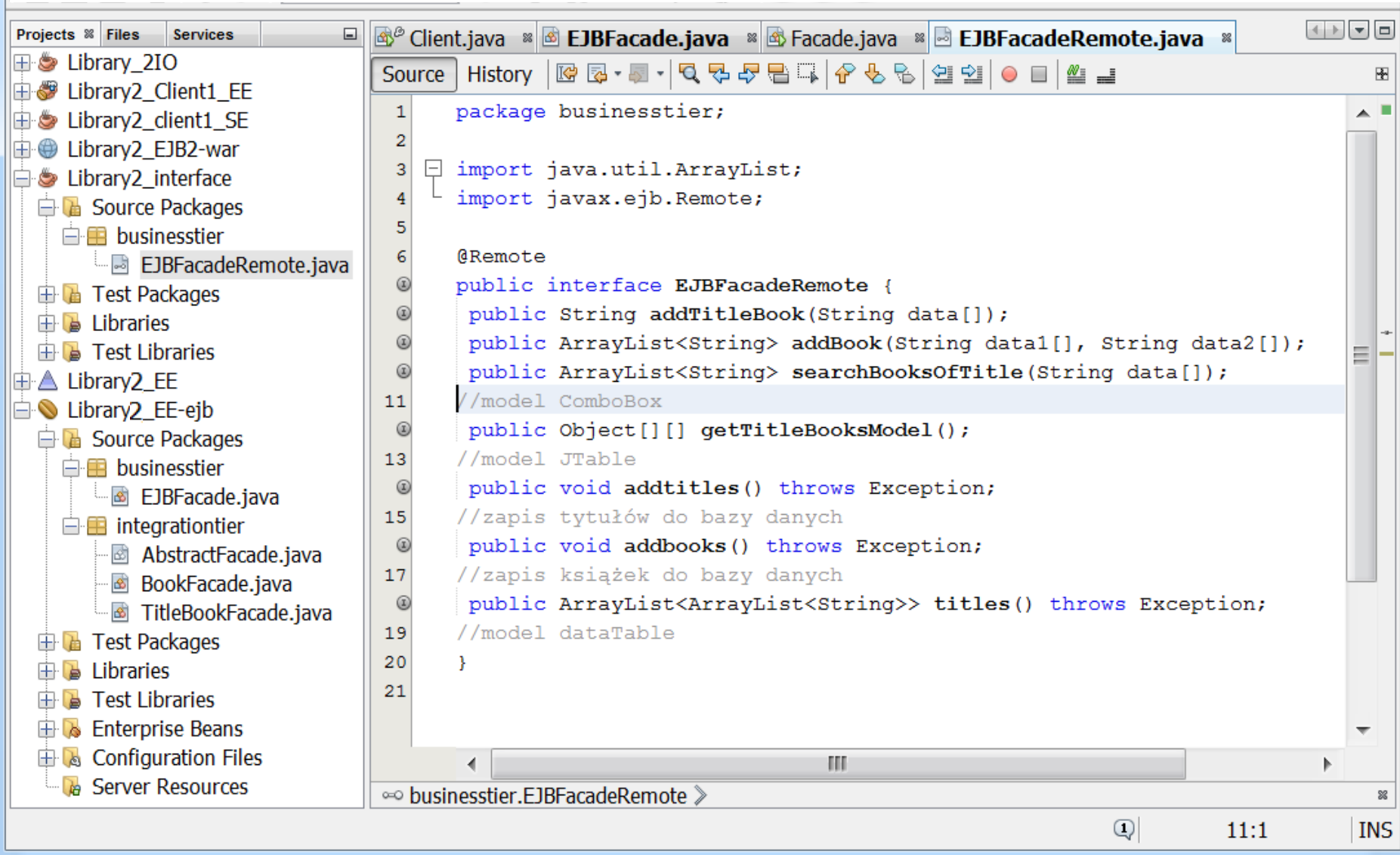
```
1 package integrationtier;
2 import ...6 lines
3
4
5
6
7
8
9 @Stateless
10 public class TitleBookFacade extends AbstractFacade<TitleBook> {
11
12     @PersistenceContext(unitName = "Library3_EE-ejbPU")
13     private EntityManager em;
14
15     @Override
16     protected EntityManager getEntityManager() {...3 lines}
17
18     public TitleBookFacade() {...3 lines}
19
20     public void addTitleBooks(List<TitleBook> titles) {
21         for (TitleBook title: titles)
22             if (title.getId() == null)
23                 getEntityManager().persist(title);
24     }
25 }
26
27
```
- Bottom Bar:** Shows the current file "integrationtier.TitleBookFacade" and the method "addTitleBooks".
- Status Bar (Bottom Right):** Displays "26:5" and "INS".

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



```
7   import subbusinessstier.entities.Book;
8   import subbusinessstier.entities.TitleBook;
9
10  @Stateless
11  public class BookFacade extends AbstractFacade<Book> {
12
13      @PersistenceContext(unitName = "Library3_EE-ejbPU")
14      private EntityManager em;
15
16      @Override
17      protected EntityManager getEntityManager() {...3 lines }
18
19      public BookFacade() {...3 lines }
20
21      public void addBooks(List<TitleBook> titles) {
22          for (TitleBook title:titles) {
23              if (title.getId() == null)
24                  continue;
25              for(Book book:title.getBooks())
26                  if (book.getId() == null)
27                      getEntityManager().persist(book);
28          }
29      }
30  }
31
32 }
```

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



The screenshot displays an IDE window with the following components:

- Projects Panel (Left):** Shows a project structure with source packages. The package `businessstier` is expanded, showing the file `EJBFacadeRemote.java`.
- Source Editor (Right):** Displays the source code for `EJBFacadeRemote.java`. The code is as follows:

```
1 package businessstier;
2
3 import java.util.ArrayList;
4 import javax.ejb.Remote;
5
6 @Remote
7 public interface EJBFacadeRemote {
8     public String addTitleBook(String data[]);
9     public ArrayList<String> addBook(String data1[], String data2[]);
10    public ArrayList<String> searchBooksOfTitle(String data[]);
11    //model ComboBox
12    public Object[][] getTitleBooksModel();
13    //model jTable
14    public void addtitles() throws Exception;
15    //zapis tytułów do bazy danych
16    public void addbooks() throws Exception;
17    //zapis książek do bazy danych
18    public ArrayList<ArrayList<String>> titles() throws Exception;
19    //model dataTable
20 }
21
```
- Bottom Panel:** Shows the package path `businessstier.EJBFacadeRemote` and the current time `11:1`.

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

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

The screenshot shows the NetBeans IDE 8.2 interface. The title bar reads "Library3_EE - NetBeans IDE 8.2". The menu bar includes "File", "Edit", "View", "Navigate", "Source", "Refactor", "Run", "Debug", "Profile", "Team", "Tools", "Window", and "Help". A search bar is present with the text "Search (Ctrl+I)".

The left sidebar shows a project tree with the following items:

- Library_2IO
- Library2_Client1_EE
- Library2_client1_SE
- Library2_EJB2-war
- Library2_interface
- Library3_EE (selected)
- Library3_EE

The main editor window displays the source code of "BookFacade.java":

```
7 import subbusinessstier.entities.Book;
8 import subbusinessstier.entities.TitleBook;
9
10 @Stateless
11 public class BookFacade extends AbstractFacade<Book> {
12     private EntityManager em;
13     private String unitName = "Library3_EE-ejbPU";
14     private EntityManagerAGER em;
15
16     public BookFacade() {
17         super();
18     }
19
20     public void addBooks() {
21         // ...
22     }
23 }
```

The bottom console window shows the output of the GlassFish Server:

```
GlassFish Server * Library3_EE (run-deploy) *
names for EJB TitleBookFacade: [java:global/Libr
names for EJB BookFacade: [java:global/Library3_
05
Observer method [BackedAnnotatedMethod] org.glass
Observer method [BackedAnnotatedMethod] public or
Observer method [BackedAnnotatedMethod] private o
as successfully deployed in 4 747 milliseconds.
```

6. Wygenerowanie pustych tabel w bazie danych

The screenshot displays the NetBeans IDE 8.2 interface. The left pane shows a project named 'jdbc:derby://localhost:1527/Biblioteka [Biblioteka on BIOBLIOTEKA]'. Under the 'BIOBLIOTEKA' schema, there is a 'Tables' folder containing 'BOOK' and 'TITLEBOOK'. The 'BOOK' table has columns: ID, NUMBER, and TITLEBOOK_ID. The 'TITLEBOOK' table has columns: ID, DTYPE, ISBN, AUTHOR, PUBLISHER, TITLE, and ACTOR. There are also 'Indexes', 'Foreign Keys', and 'Views' folders. The right pane shows the 'Client.java' source file with the following code:

```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package client;
```

The bottom right pane shows the 'Output' window with the following text:

```
Java DB Database Process
pre-run-deploy:
Initial deploying I
Completed initial c
post-run-deploy:
run-deploy:
BUILD SUCCESSFUL (t
```

The bottom status bar shows '<No View Available>' and '1:1 INS'.

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. Przykład modelowania i projektowania części **warstwy biznesowej** z obiektami typu POJO. Wykonanie aplikacji dwuwarstwowej.
5. Przykłady architektury wielowarstwowej aplikacji internetowej typu EE. Wykonanie aplikacji typu EE. **Warstwa biznesowa**: komponenty typu EJB + obiekty POJO
6. **Warstwa zasobów (EIS)**- baza danych w systemie baz danych Derby
7. Utworzenie obiektowego modelu danych do utrwalania ORM
8. **Warstwa integracji**. Zastosowanie wzorca projektowego typu *Domain Store* w technologii JPA (Java Persistence) na platformie Java EE
9. **Warstwa prezentacji - JSF**

1. Wykonanie projektu typu Web Application i zdefiniowanie formularzy JSF

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes:

- Library_2IO
- Library2_Client1_EE
- Library2_client1_SE
- Library2_EJB2-war
- Web Pages (highlighted with a red box)
 - WEB-INF
 - web.xml
 - presentation_tier_view
 - Show_data.xhtml
 - Store_data.xhtml
 - resources
 - index2.xhtml
 - template.xhtml
- Source Packages
 - presentation_tier
 - Managed_Bean1.java
- Test Packages
- Libraries
- Test Libraries
- Configuration Files
- Library2_interface
- Library2_EE
- Library2_EE-ejb

The code editor displays the content of `template.xhtml` (file: `TitleBookFacade.java`):

```
<body>
  <ui:composition template="../../../template.xhtml">
    <ui:define name="content">
      <h:form styleClass="jsfcrud_list_form">
        <h:panelGroup ...3 lines />
        <h:outputText escape="false" value="Lista produktow pusta"
          rendered="#{managed_Bean1.items.rowCount == 0}"/>
        <h:panelGroup rendered="#{managed_Bean1.items.rowCount > 0}">
          <h:dataTable value="#{managed_Bean1.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 ...3 lines />
              <h:outputText value="#{item.get(0)}"/>
            </h:column>
            <h:column> <f:facet ...3 lines />
              <h:outputText value="#{item.get(1)}"/>
            </h:column>
            <h:column> <f:facet ...3 lines />
              <h:outputText value="#{item.get(2)}"/>
            </h:column>
            <h:column> <f:facet ...3 lines />
              <h:outputText value="#{item.get(3)}"/>
            </h:column>
            <h:column> <f:facet ...3 lines />
              <h:outputText value="#{item.get(4)}"/>
            </h:column>
          </h:dataTable>
        </h:panelGroup>
      </h:form>
    </ui:define>
  </ui:composition>
</body>
</html>
```

2. Definiowanie formularzy JSF (cd)

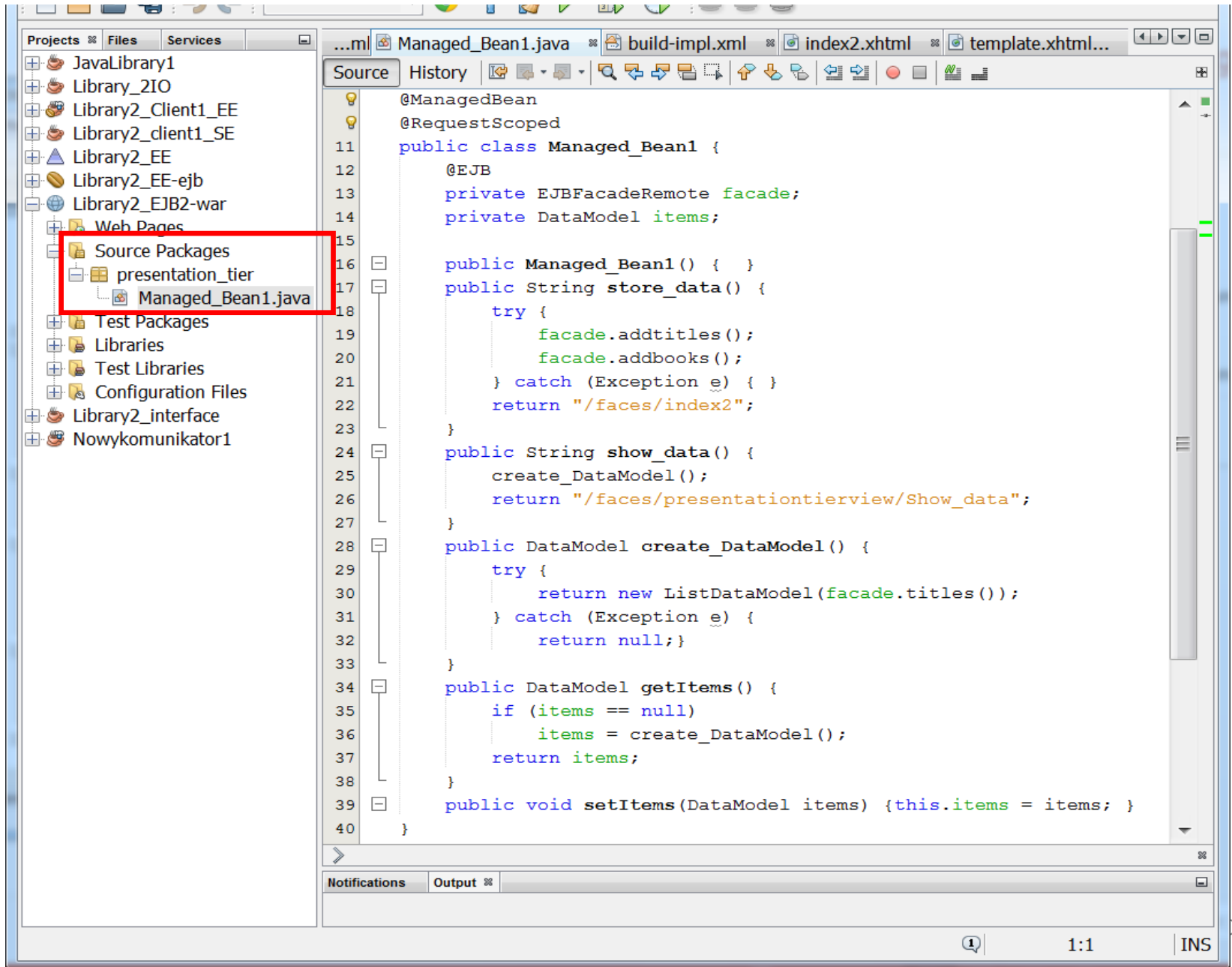
The screenshot displays the NetBeans IDE 8.2 interface for a project named "Library2_EJB2-war". The left-hand "Projects" pane shows a tree view of the project structure. A red rectangle highlights the "Web Pages" folder, which contains a "WEB-INF" subfolder. Inside "WEB-INF", there is a "web.xml" file and a "presentation_tier_view" folder. The "presentation_tier_view" folder contains two XHTML files: "Show_data.xhtml" and "Store_data.xhtml".

The main editor window shows the source code of "Show_data.xhtml". The code is as follows:

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4       xmlns:ui="http://java.sun.com/jsf/facelets"
5       xmlns:h="http://java.sun.com/jsf/html">
6
7     <body>
8         <ui:composition template="../../template.xhtml">
9             <ui:define name="content">
10                 <h:form>
11                     <h:commandButton action="#{managed_Bean1.store_data}"
12                                     value="Store data"/><br/>
13                 </h:form>
14             </ui:define>
15         </ui:composition>
16
17     </body>
18 </html>
```

The IDE interface includes a menu bar (File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help), a search bar (Search (Ctrl+I)), and a toolbar with various icons for file operations and development. The status bar at the bottom shows "1:1" and "INS".

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



The screenshot displays an IDE window with the following components:

- Project Explorer (Left):** Shows a project structure with 'Source Packages' containing 'presentation_tier' and 'Managed_Bean1.java' (highlighted with a red box).
- Source Editor (Center):** Displays the source code for `Managed_Bean1.java`. The code is as follows:

```
@ManagedBean
@RequestScoped
public class Managed_Bean1 {
    @EJB
    private EJBFacadeRemote facade;
    private DataModel items;

    public Managed_Bean1() { }
    public String store_data() {
        try {
            facade.addtitles();
            facade.addbooks();
        } catch (Exception e) { }
        return "/faces/index2";
    }

    public String show_data() {
        create_DataModel();
        return "/faces/presentationtier/Show_data";
    }

    public DataModel create_DataModel() {
        try {
            return new ListDataModel(facade.titles());
        } catch (Exception e) {
            return null;
        }
    }

    public DataModel getItems() {
        if (items == null)
            items = create_DataModel();
        return items;
    }

    public void setItems(DataModel items) {this.items = items; }
}
```
- Bottom Panel:** Contains 'Notifications' and 'Output' tabs.
- Status Bar (Bottom Right):** Shows '1:1' and 'INS'.

4. Uruchomienie bazodanowej aplikacji typu Enterprise na platformie JavaEE zawierającej dwa typy aplikacji klienckiej: desktopową i internetową

The screenshot shows a desktop application window titled "MenuDemo". The window has a menu bar with "A Menu" and "Another Menu". The main content area contains a form with the following fields:

- Title: Tytul1
- Author: Autor1
- ISBN: 123456
- Publisher: Wydawca1
- Actor: (empty)

Below the form is a button labeled "Add title".

The screenshot shows a desktop application window titled "MenuDemo". The window has a menu bar with "A Menu" and "Another Menu". The main content area contains a table with the following data:

Publisher	ISBN	Title	Author	Actor
Wydawca1	123456	Tytul1	Autor1	

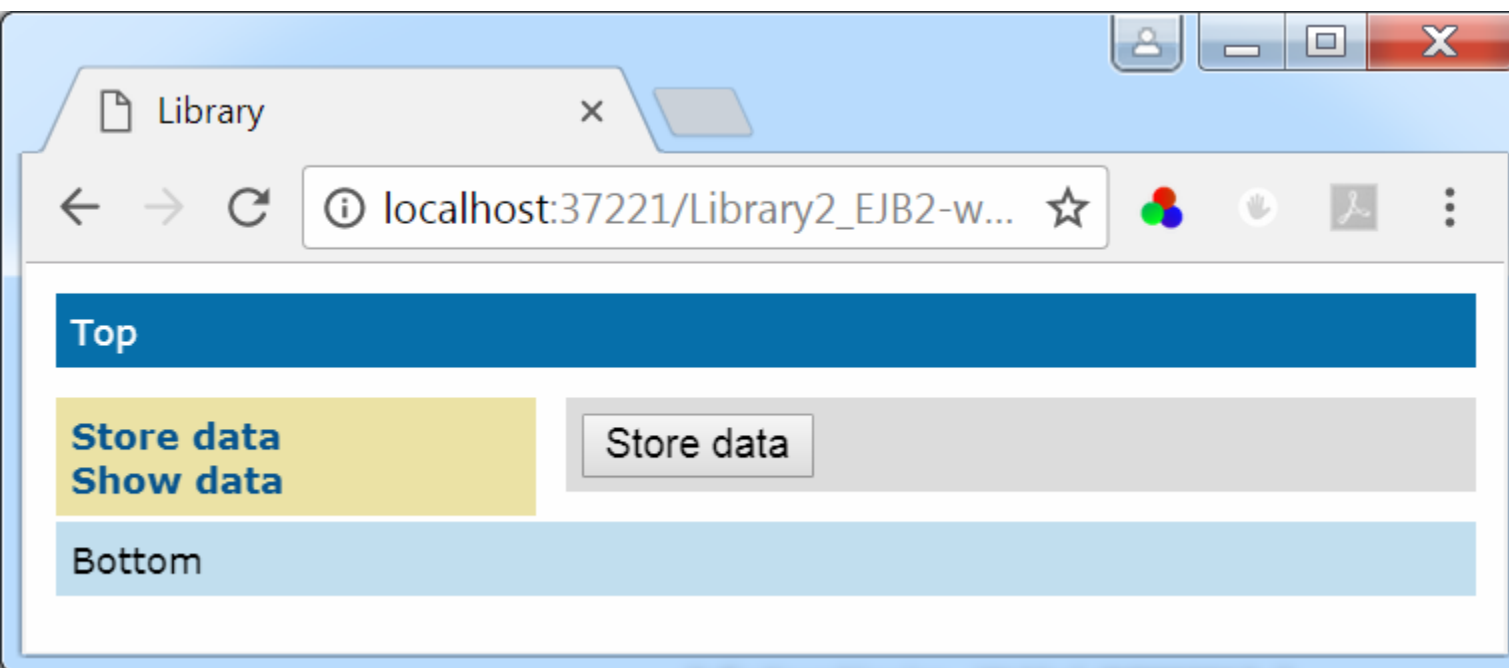
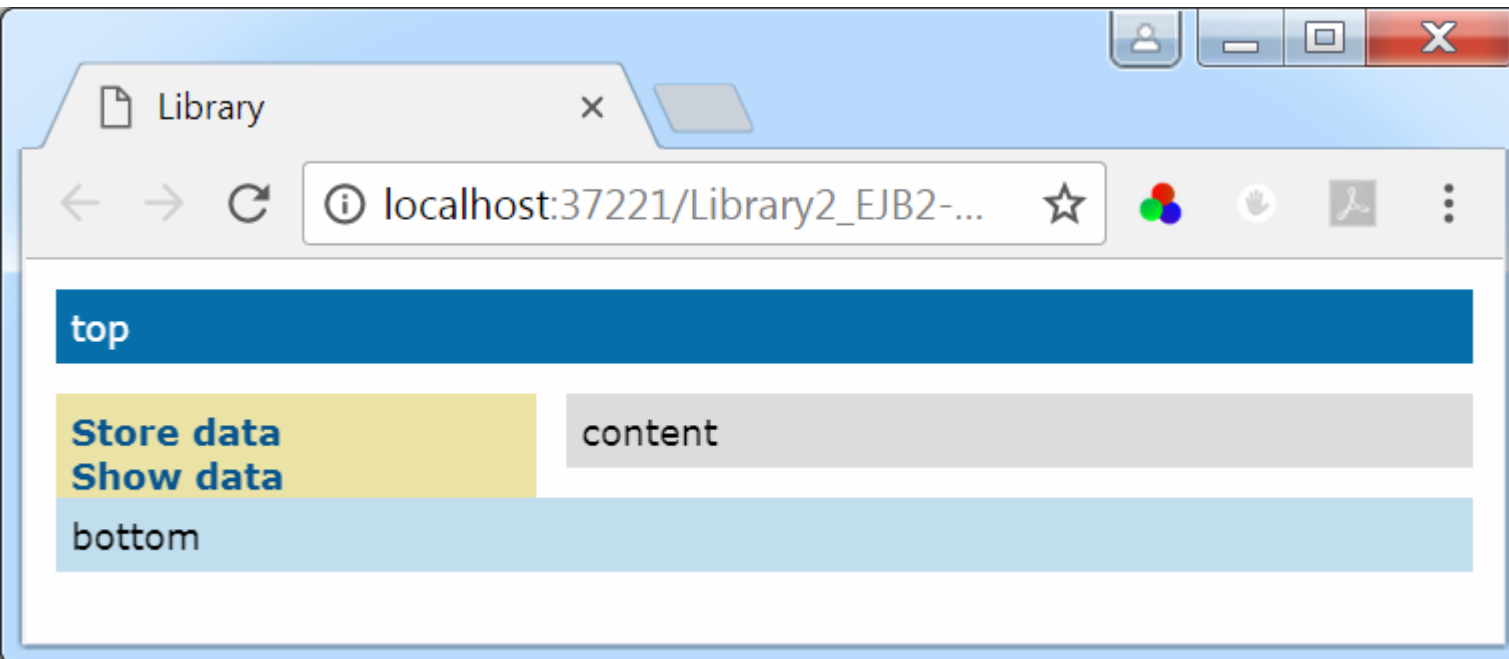
Below the table is a form with the following fields:

- Number of a book: 1

Below the form are two buttons: "Add book" and "Clear selection".

At the bottom of the window is a label "Books" and a dropdown menu showing the text: "Title: Tytul1 Author: Autor1 ISBN: 123456 Publisher: Wydawca1 Number: 1".

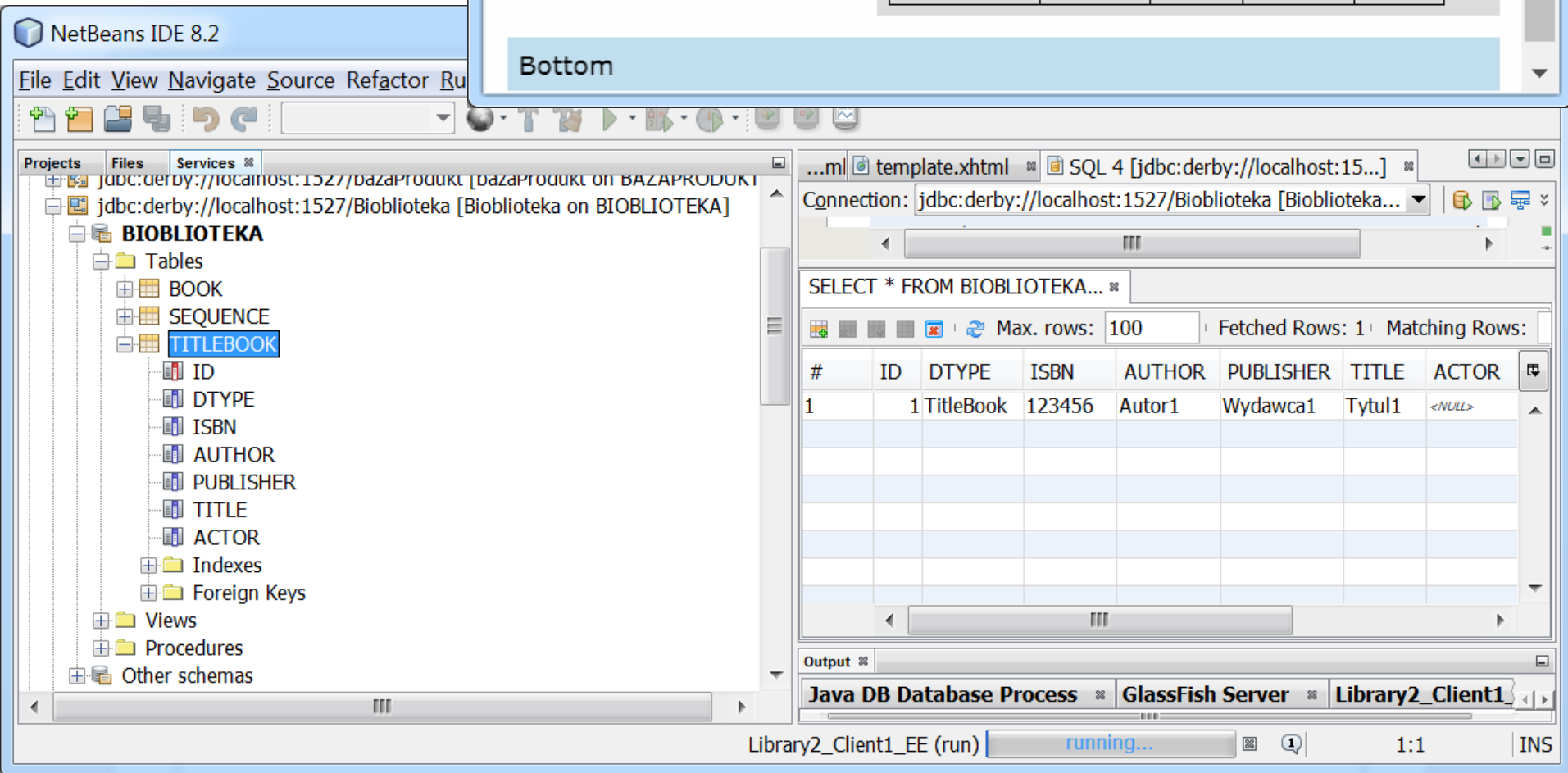
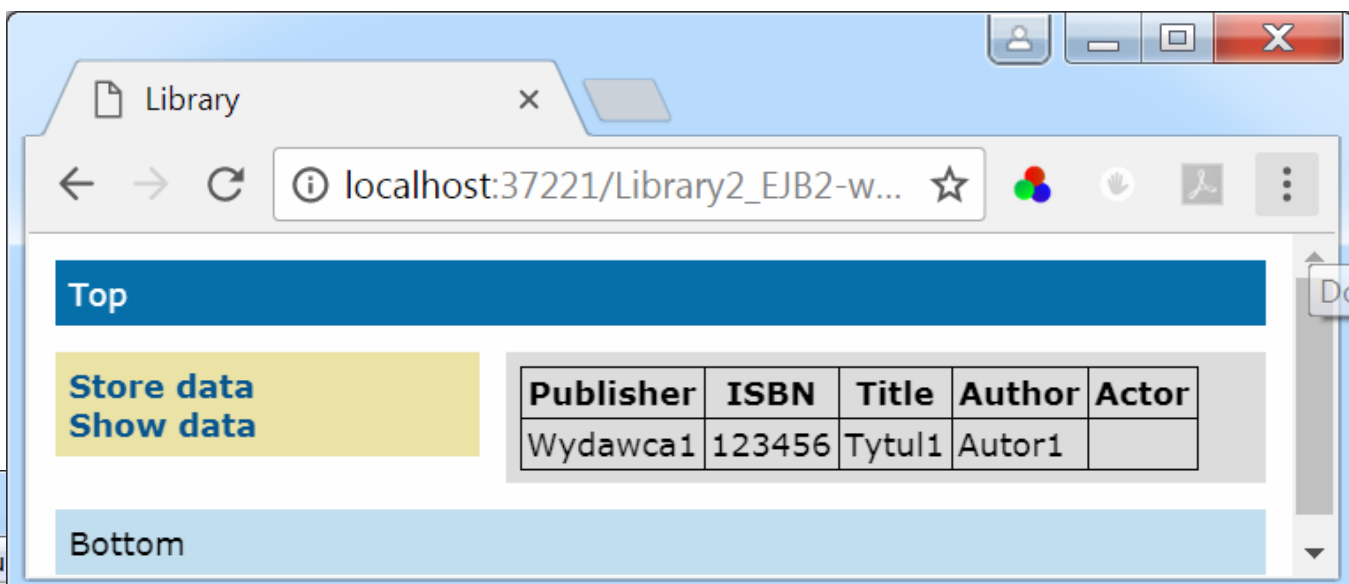
5. Uruchomiana aplikacja internetowa – formularz **Store data** służy do zapisania danych tytułów i książek do bazy danych



6. Uruchomiana aplikacja internetowa – formularz

Show data

służy do wyświetlenia danych tytułów pobranych z bazy danych



Zagadnienia

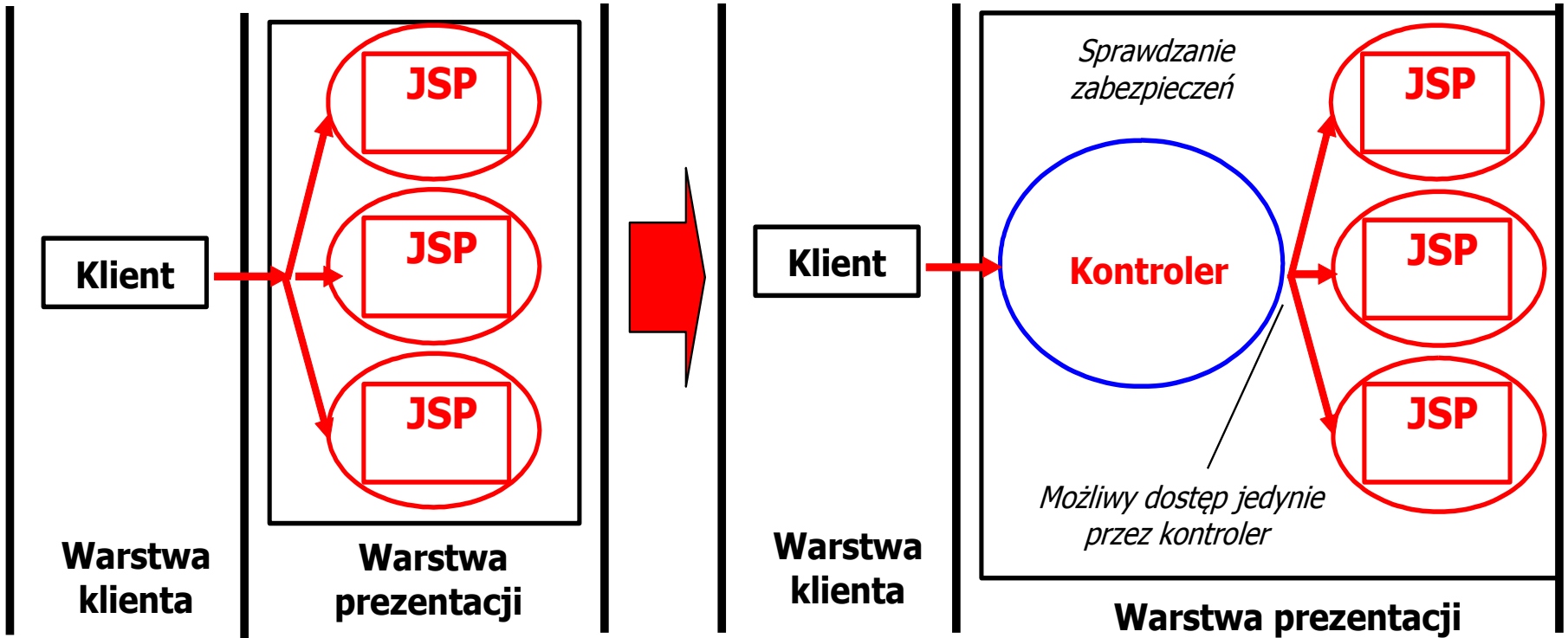
1. Wielowarstwowa architektura systemu informatycznego
2. Ocena i poprawa (refaktoryzacja) architektury wielowarstwowej systemu informatycznego
3. Wzorce projektowe stosowane przy budowie wielowarstwowej aplikacji internetowej
4. Przykład modelowania i projektowania części **warstwy biznesowej** z obiektami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

Dodatek

- 1. Refaktoryzacja warstwy prezentacji**
- 2. Refaktoryzacja warstwy biznesowej**
- 3. Tworzenie warstwy integracji**

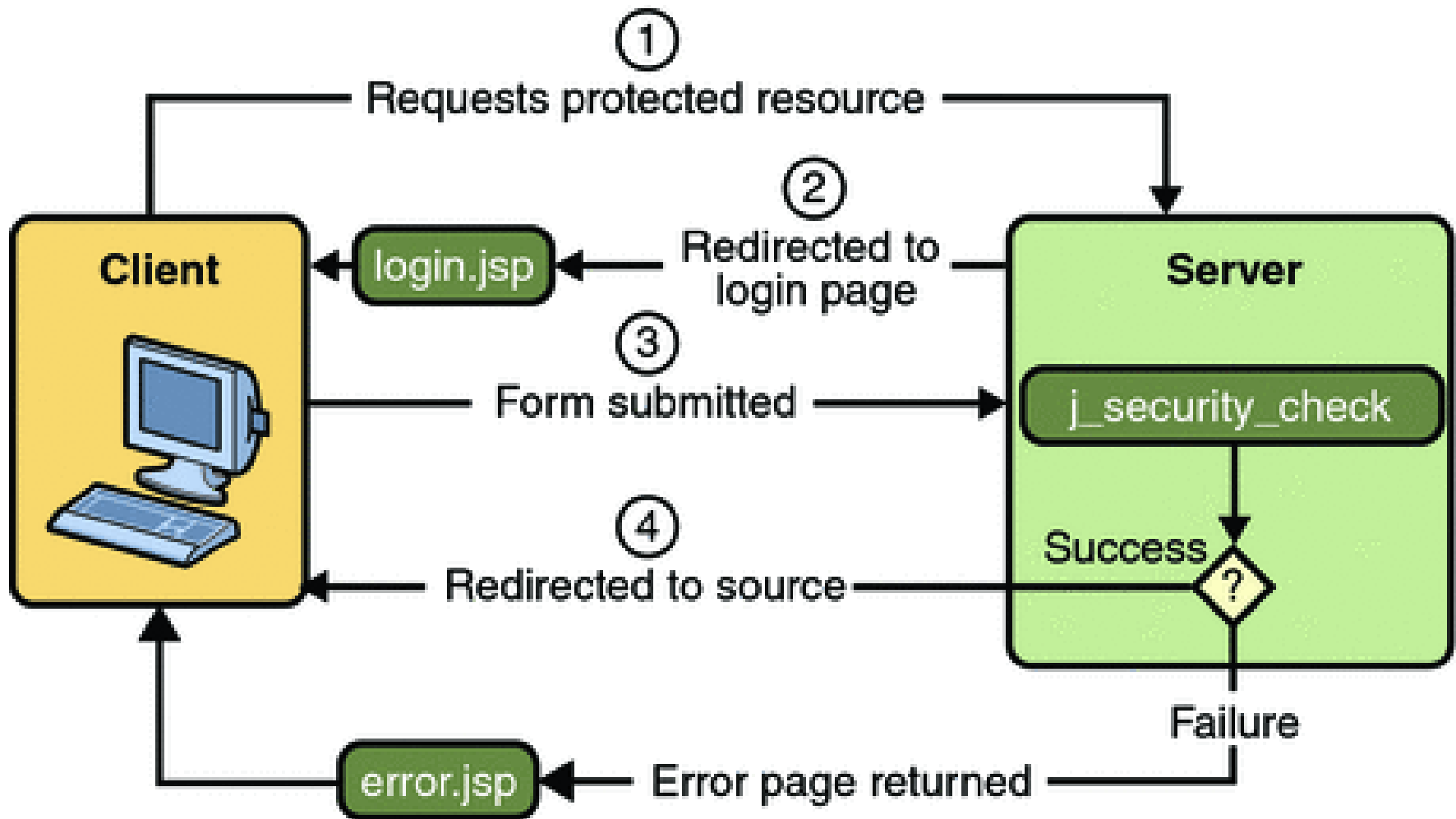
1. Refaktoryzacja warstwy prezentacji

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



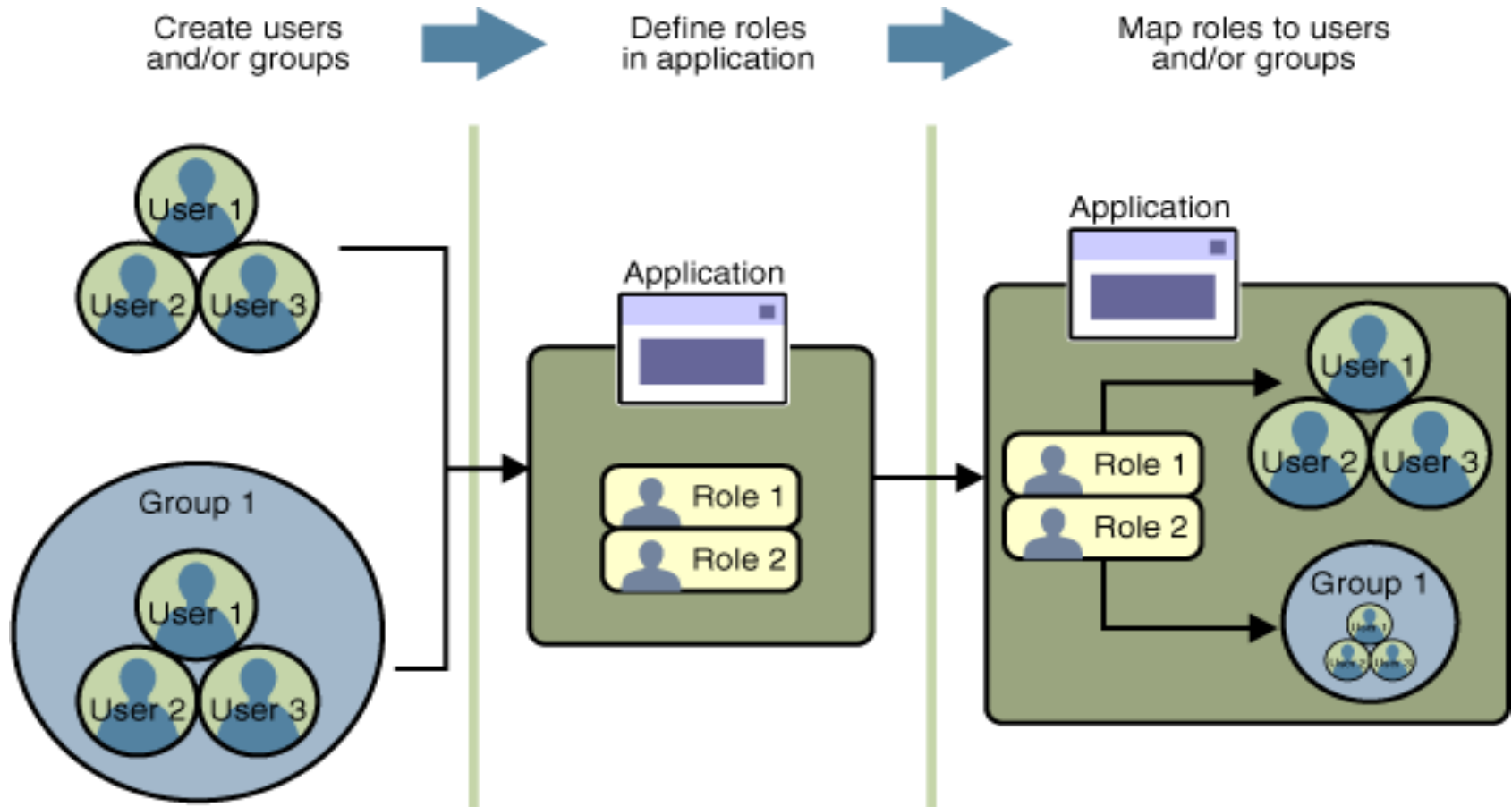
Przebieg uwierzytelniania (logowania)

<http://download.oracle.com/javaee/5/tutorial/doc/bncbe.html>



Bazy użytkowników i grup, Użytkownik, Grupa, Rola

<http://download.oracle.com/javaee/5/tutorial/doc/bnbxj.html>



Rodzaje mechanizmów bezpieczeństwa w kontenerach

(kod do zarządzania aplikacją przez serwer aplikacji m.in. Bezpieczeństwem aplikacji)

- **Deklaratywne mechanizmy bezpieczeństwa** – deklarowane za pomocą tzw. „*deployment descriptors*” (*deskryptory aplikacji np. web.xml* dla aplikacji typu *web*). 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.
- **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:
 - metody interfejsu EJBContext
 - metody interfejsu HttpServletRequest. Metody te pozwalają na podejmowanie decyzji biznesowych opartych na rolach bezpieczeństwa nadawcy lub zdalnego odbiorcy
- **Adnotacje lub metadane** są używane do specyfikowania informacji wewnątrz pliku z kodem klasy. Kiedy aplikacja jest uruchamiana, informacja ta jest używana lub pokrywana przez deskryptor aplikacji.

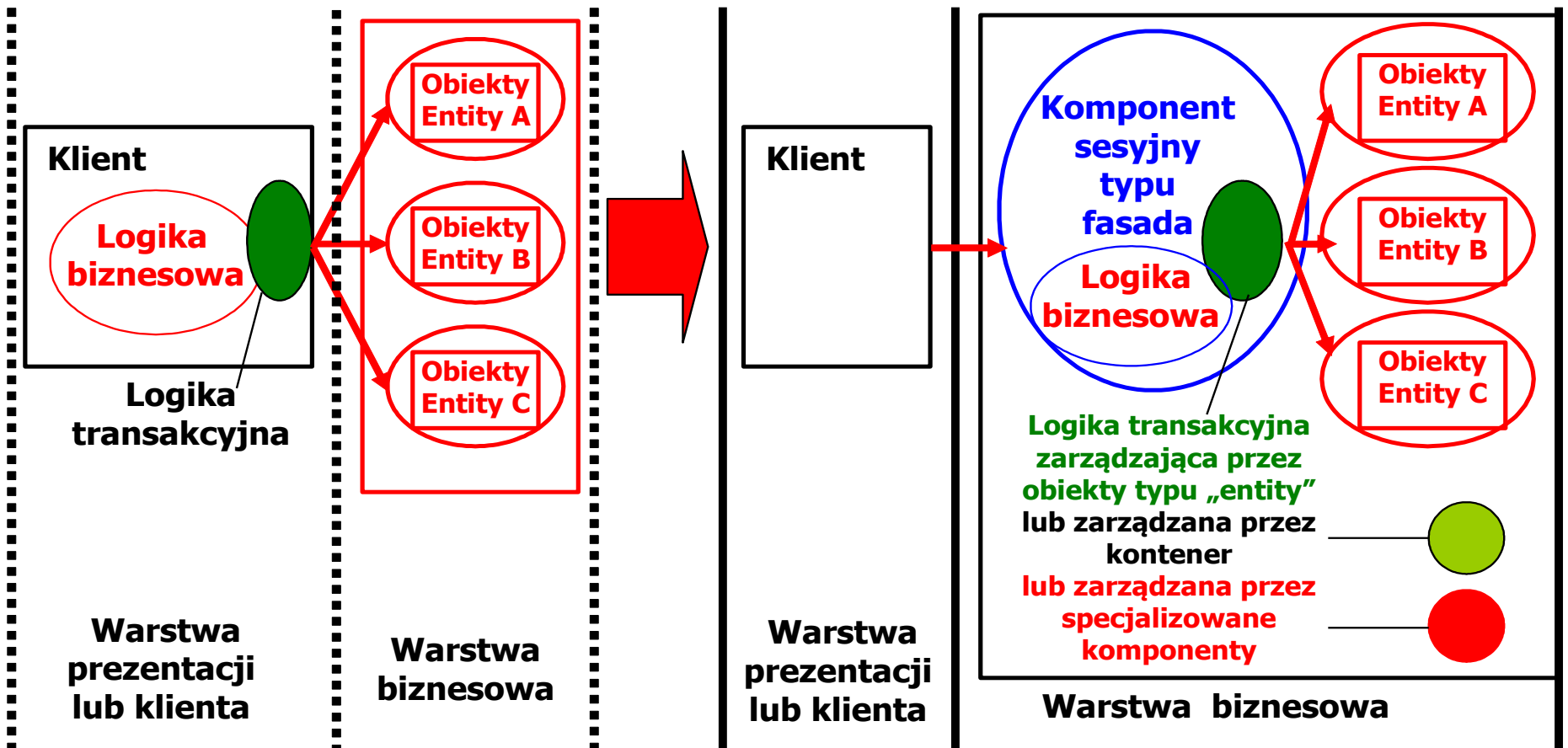
Np.

```
@DeclareRoles(„klient”)
public class Page1 extends AbstractPageBean { //... }
```

2. Refaktoryzacja warstwy biznesowej

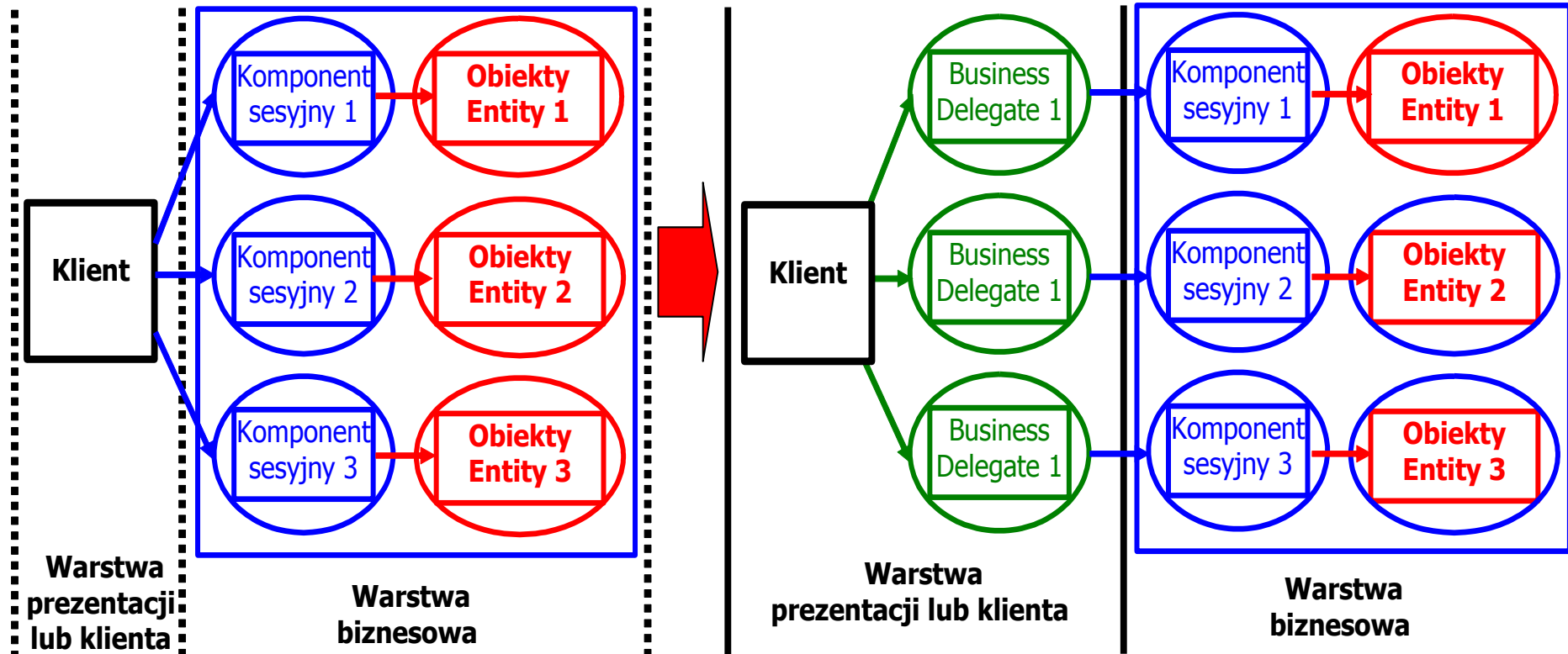
Refaktoryzacja warstwy biznesowej 1

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)**



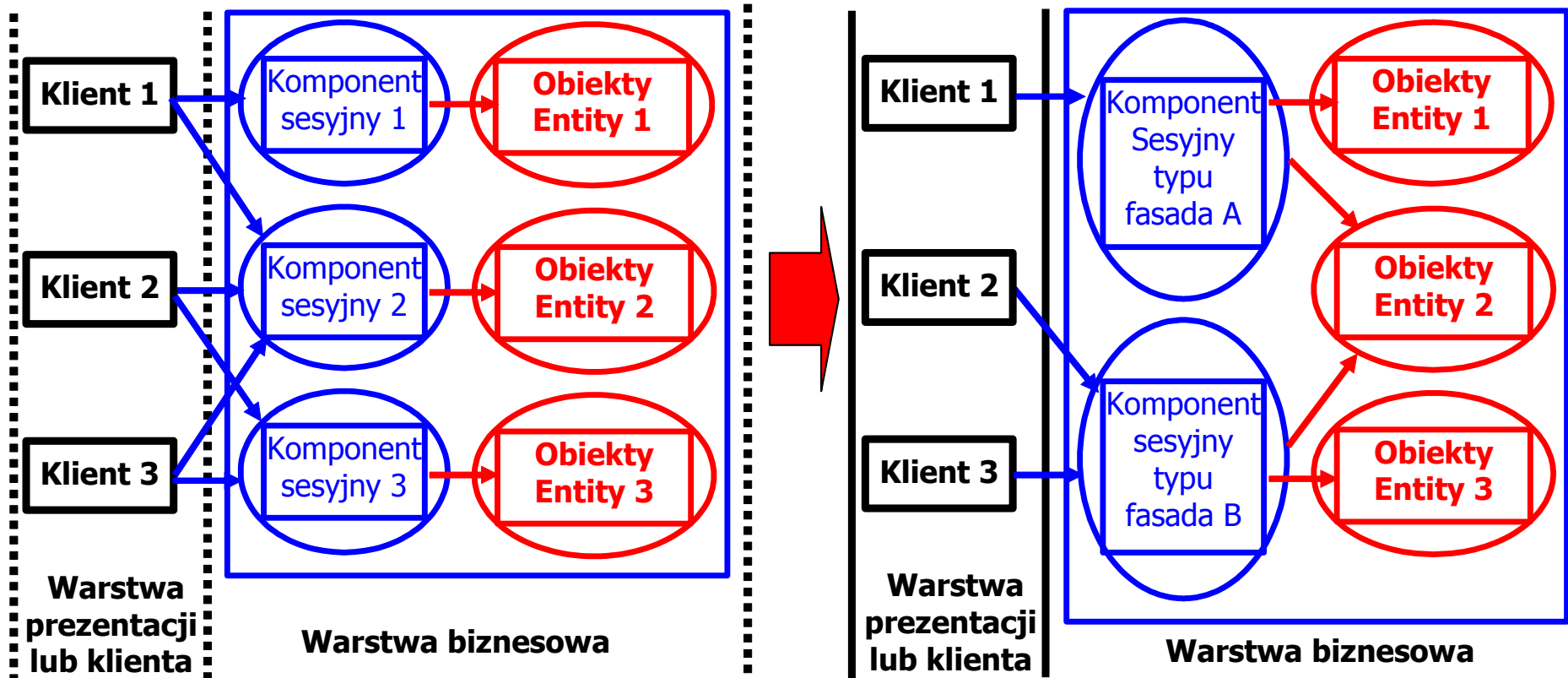
Refaktoryzacja warstwy biznesowej 2

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- komponentów **Business Delegate**)



Refaktoryzacja warstwy biznesowej 3

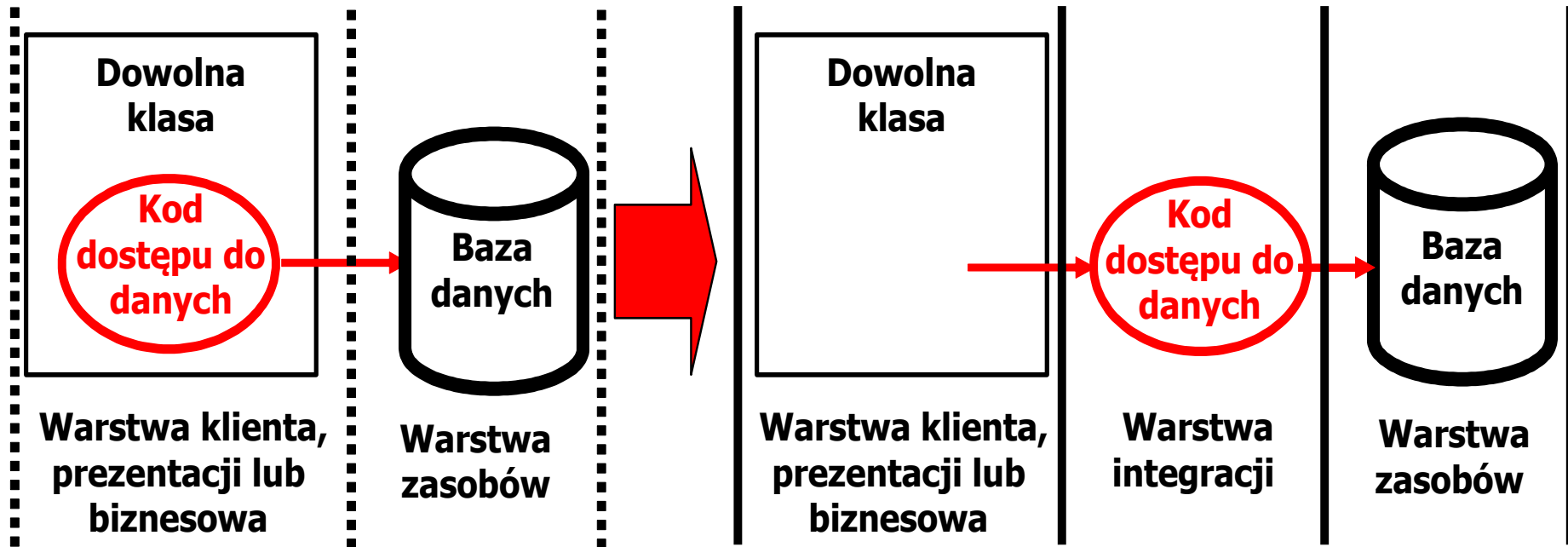
Sesyjne komponenty fasadowe typu „Control” (każdy komponent jako odrębna usługa biznesowa), hermetyzujące **obiekty danych typu „Entity”** z warstwy biznesowej są udostępniane klientom w innych warstwach. Zwykle obiekty sesyjne są jedynie pośrednikami obiektów **„Entity”**, natomiast nie hermetyzują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



3. Tworzenie warstwy integracji

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



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 i zasoby
- **Pula połączeń** kodu dostępu do danych (DAO) pozwala racjonalnie zarządzać połączeniami aplikacji z bazą danych

