

# **Projekt INP002017**

# **Instrukcja 1**

**Autor**

**Dr inż. Zofia Kruczkiewicz**

# I. Czynności wykonane zgodnie z harmonogramem grupy w tygodniach 1-6

1. Czynności (str. 3-12) wg instrukcji do lab3:

[http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab\\_INP002017\\_3.pdf](http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab_INP002017_3.pdf)

2. Czynności (str. 13-14) wg instrukcji do lab4

[http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab\\_INP002017\\_4.pdf](http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab_INP002017_4.pdf)

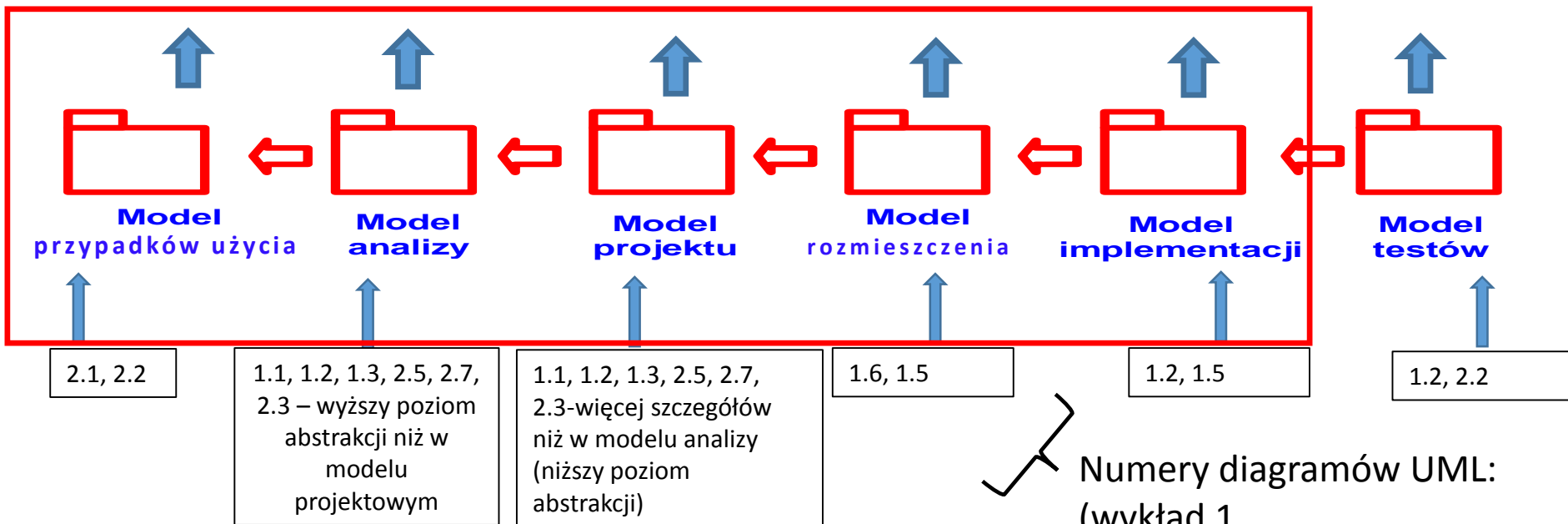
3. Czynności (str. 15-24) wg instrukcji dla lab5-7

[http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab\\_INP002017\\_5\\_1.pdf](http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab_INP002017_5_1.pdf)

[http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab\\_INP002017\\_5\\_2.pdf](http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/INP002017/Lab_INP002017_5_2.pdf)

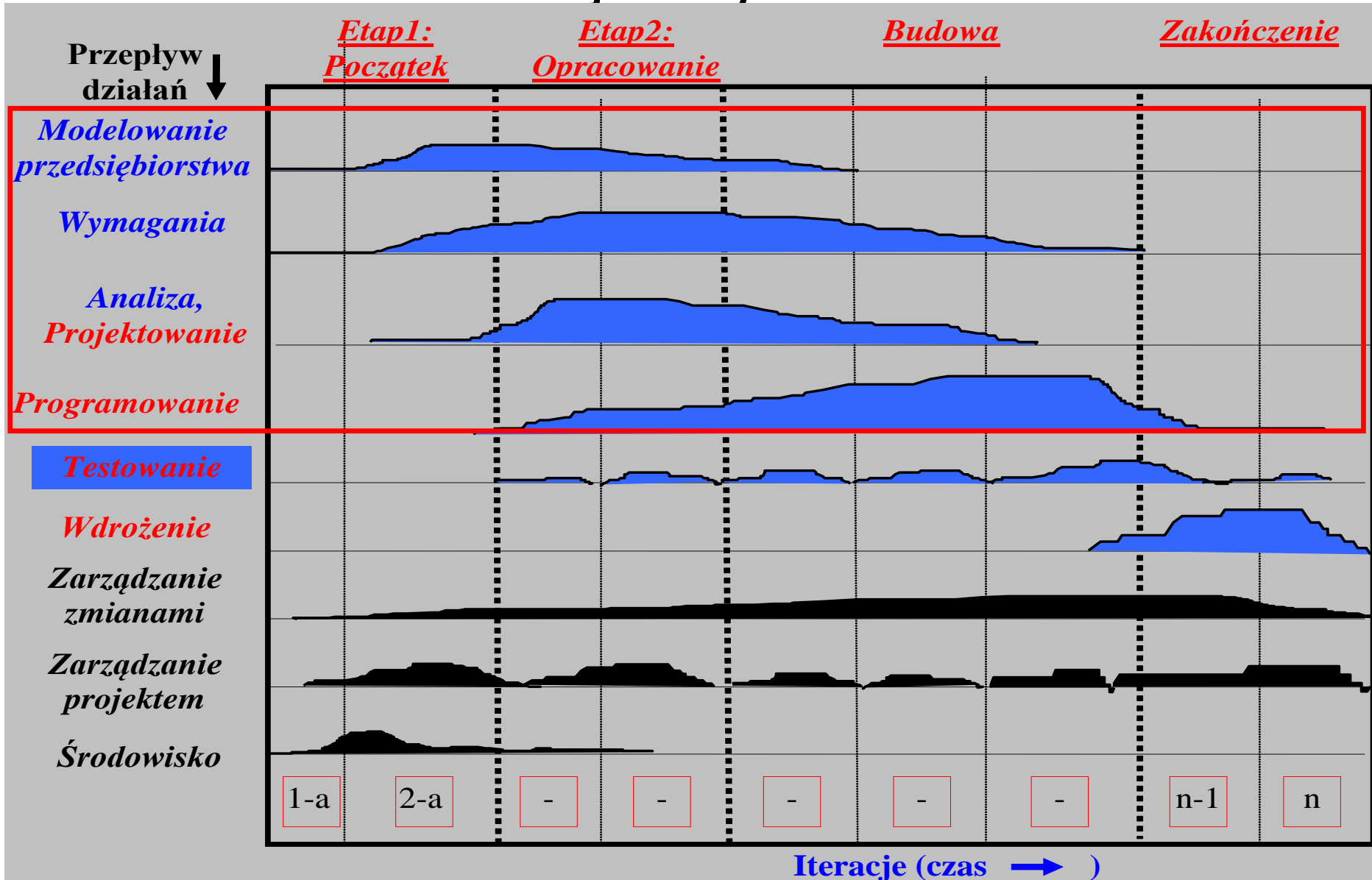
# Produkt - diagramy UML – modele, proces (wykład 1)

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



# 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 nosniku, dlatego dodatkowo tytuł zawiera dane nagrania np nazwisko aktora. Każdy egzemplarz, niezależnie, czy jest książką czy kasetą, jest opisany odrębną informacją zawierającą numer egzemplarza, który może się powtarzać dla różnych tytułów. Bibliotekarz może dodawać nowe tytuły i egzemplarze oraz je przeszukiwać, natomiast klient może jedynie przeszukiwać tytuły i sprawdzać egzemplarze wybranych tytułów.

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

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

# Opis biznesowy „świata rzeczywistego” biblioteki (cd)

## 2. Przepisy

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

## 3. Dane techniczne

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



## II. Sformułowanie wymagań funkcjonalnych i нефunkcjonalnych

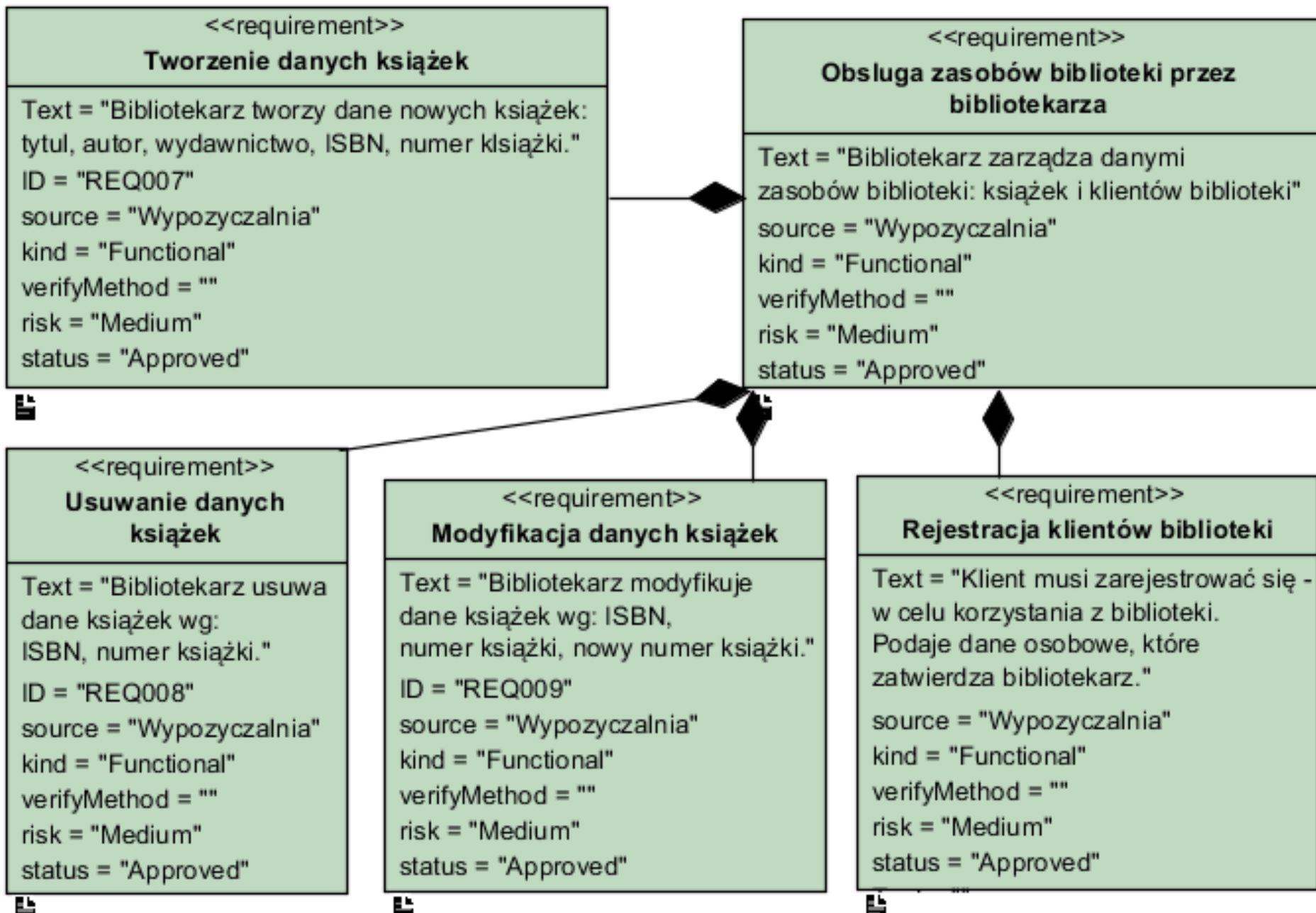
### Lista wymagań funkcjonalnych

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 umowny oraz na okres ściśle określony.

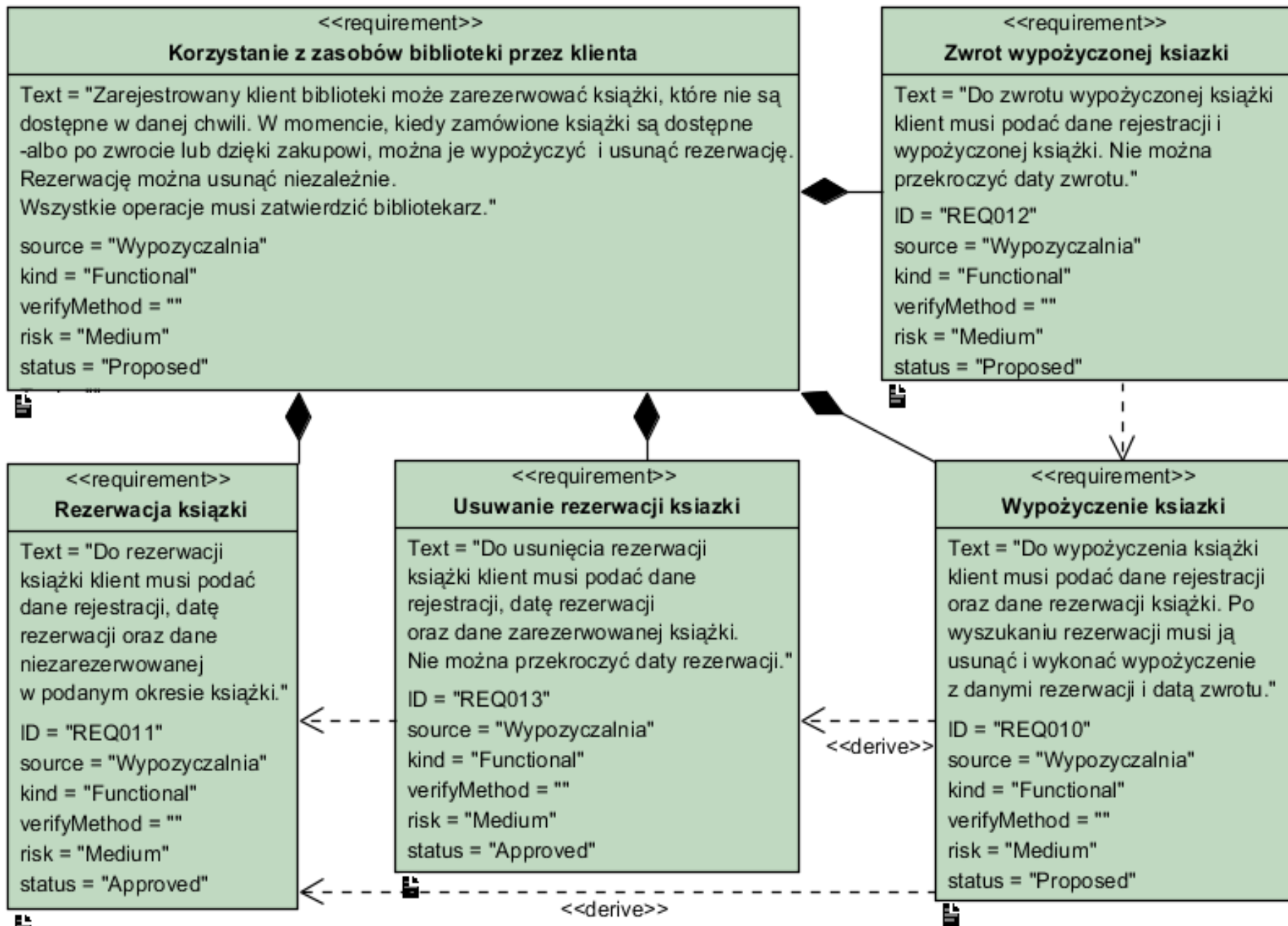
### Lista wymagań нефunkcjonalnych

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

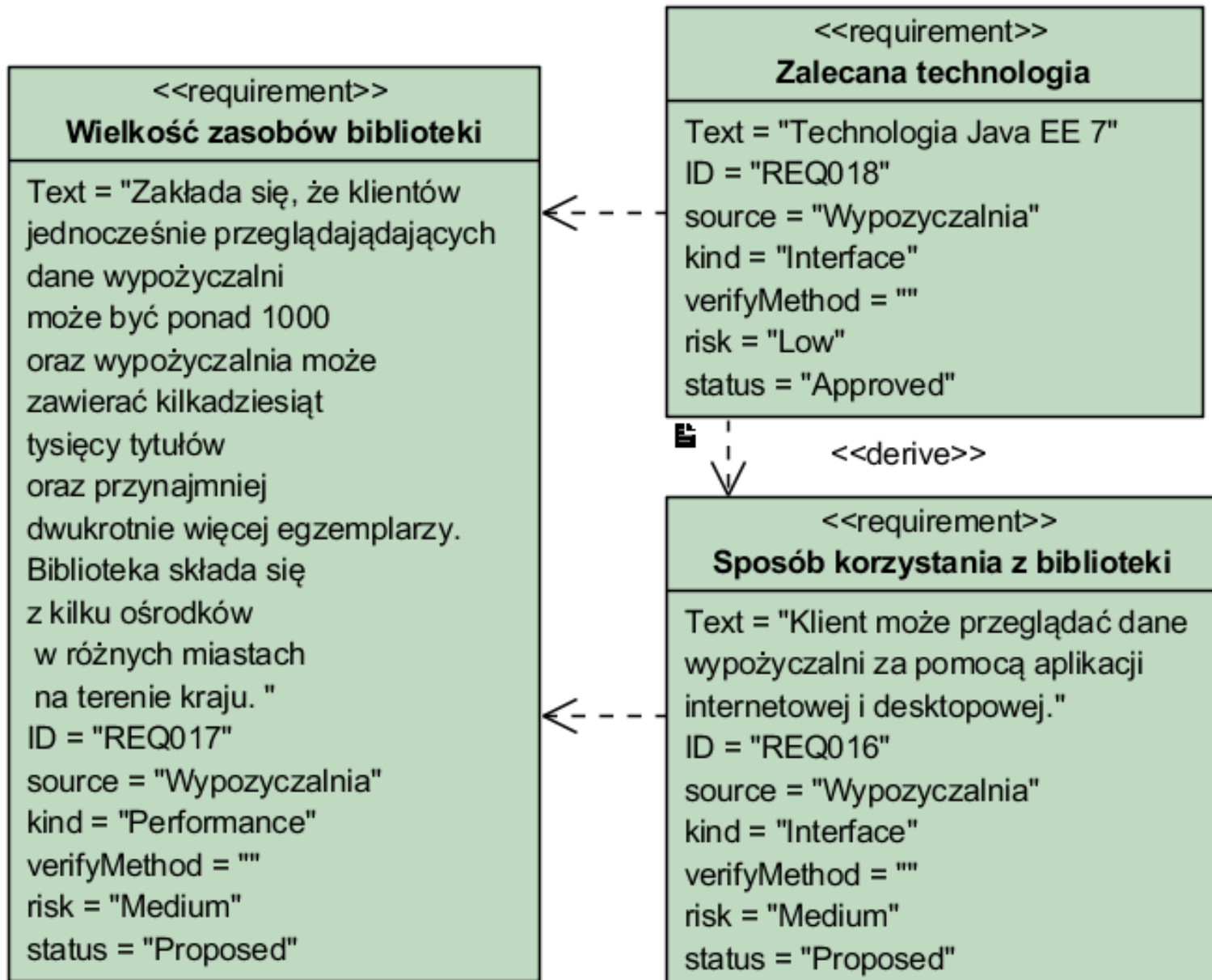
# Diagram wymagań funkcjonalnych



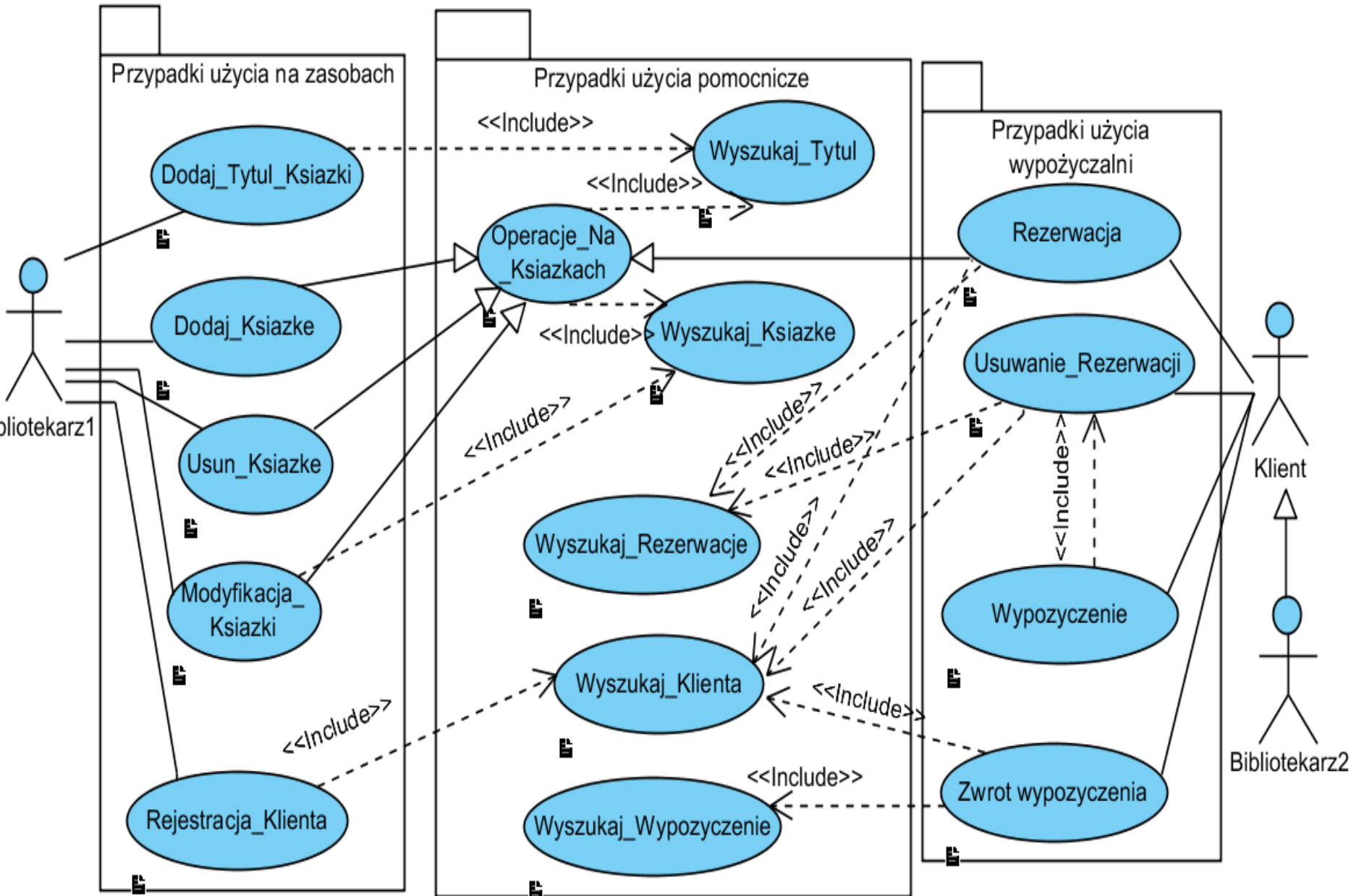
# Diagram wymagań funkcjonalnych (cd)



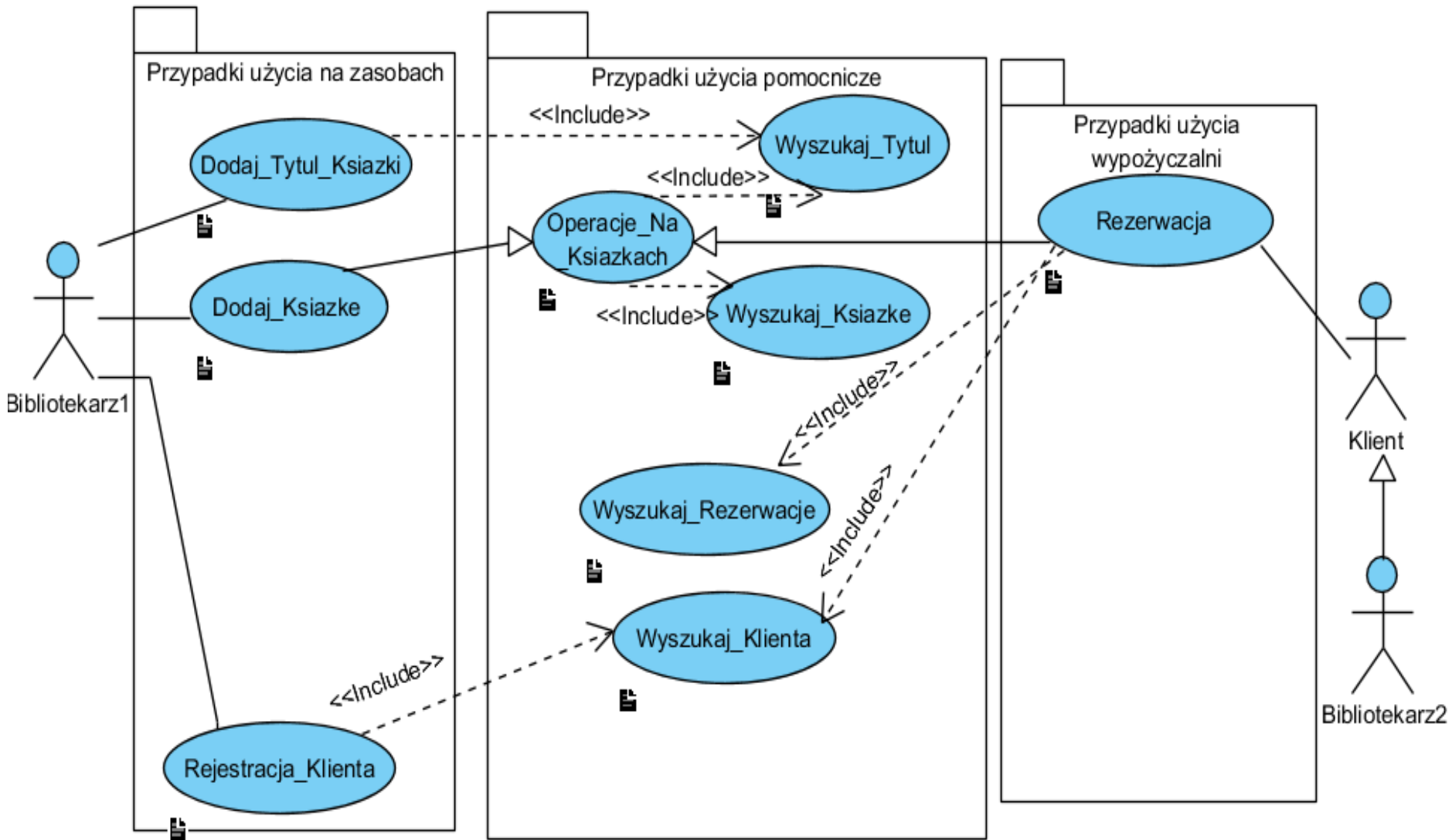
# Diagram wymagań niefunkcjonalnych (cd)



### 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**



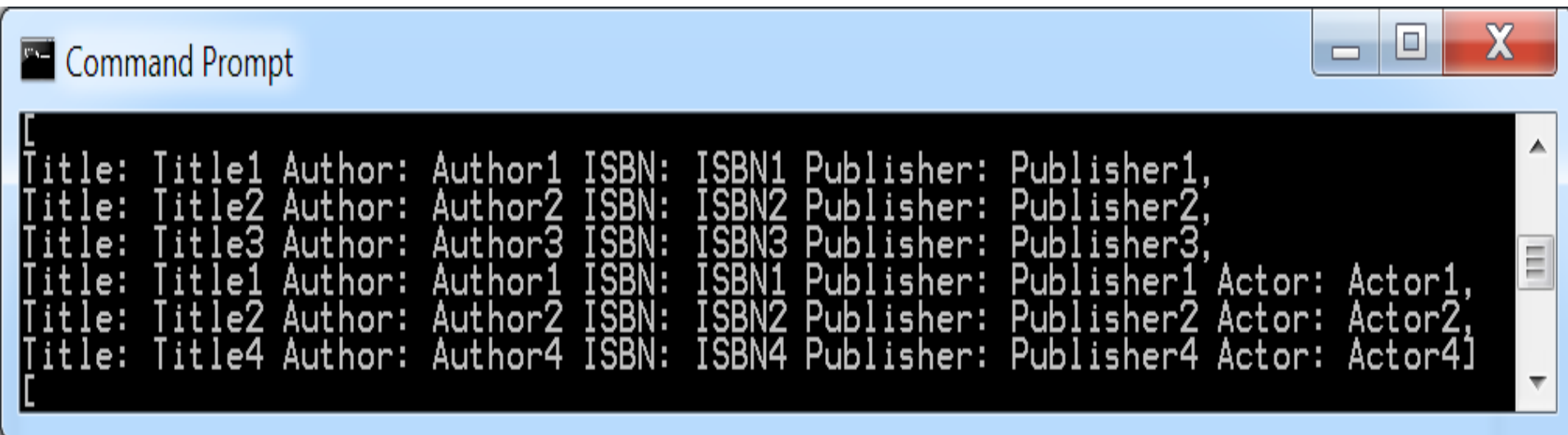
# Iteracja 1

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

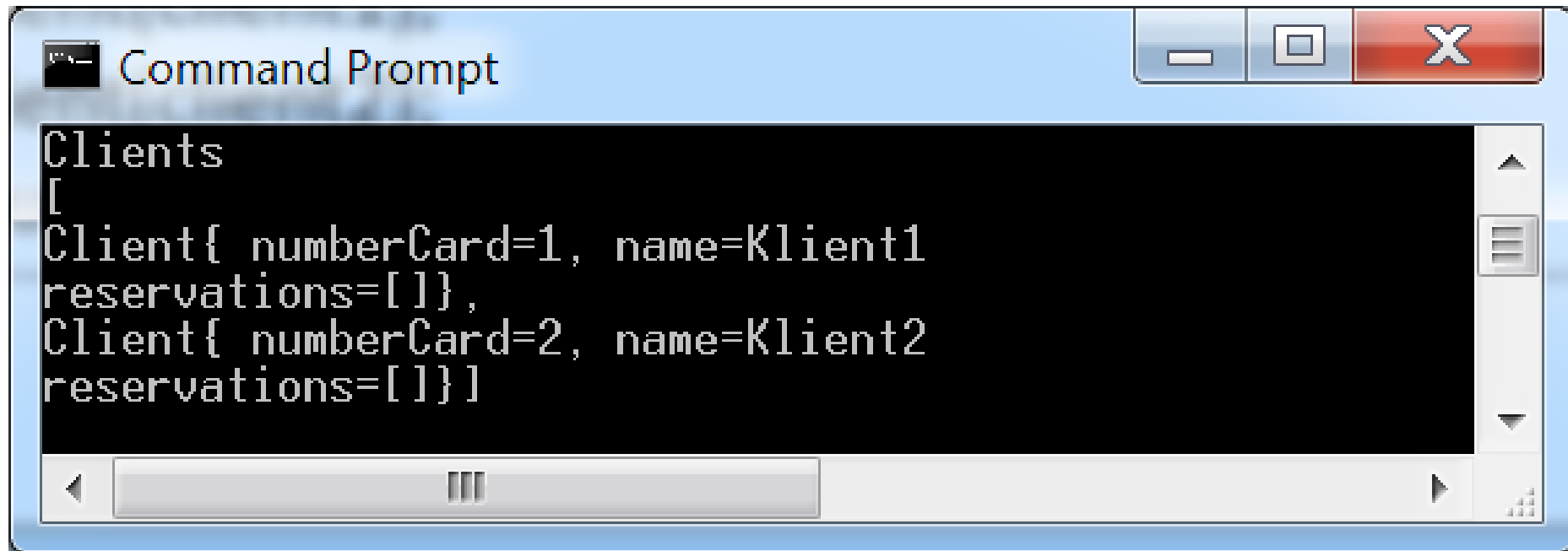
Projekt przypadku użycia

„ **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);
```



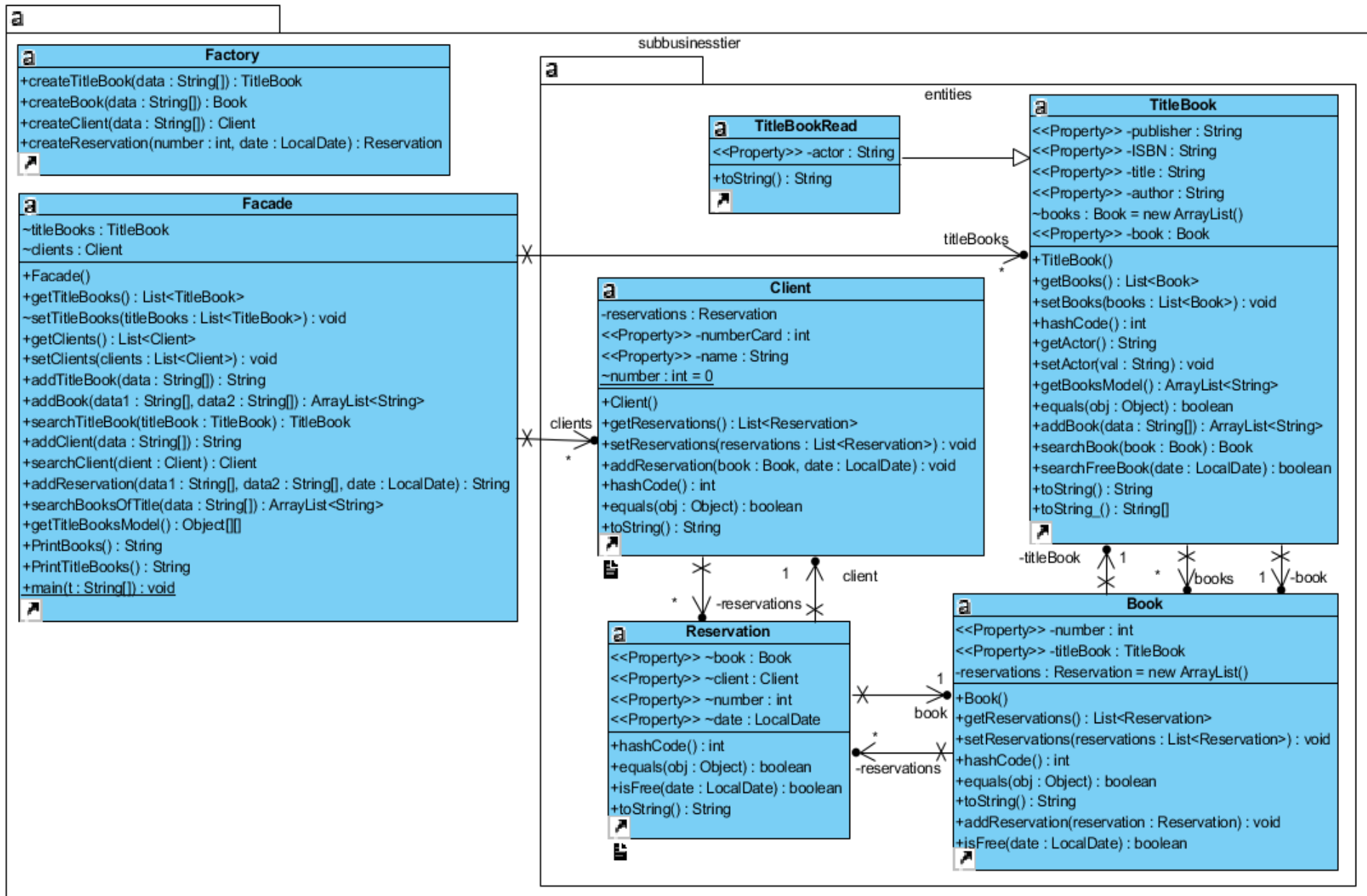
```
Command Prompt  
Clients  
[  
Client{ numberCard=1, name=Klient1  
reservations=[]},  
Client{ numberCard=2, name=Klient2  
reservations=[]}]
```

# 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}]}]
```

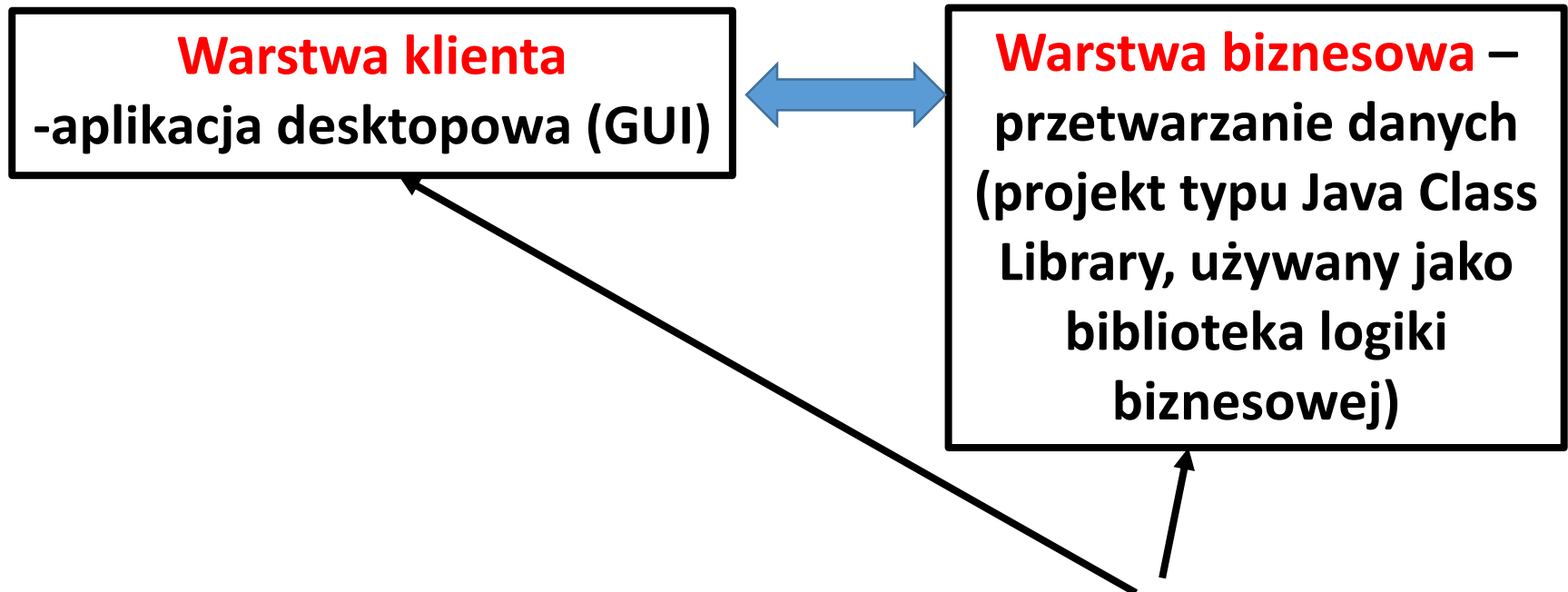


## **II. Czynności wykonane zgodnie z harmonogramem grupy w tygodniach 7-8**

**(wg materiałów do wykładu 7 Wyklad\_INP002017\_7\_2.pdf - p.4-5)**

1. Kontynuacja działań na programem z p.I (str. 26-32)- wykonanie aplikacji dwuwarstwowej obsługującej 1-go użytkownika na platformie Java SE.

# Wykonanie aplikacji dwuwarstwowej na platformie Java SE dla 1-go użytkownika, wykorzystująca kod z początkowych iteracji tworzenia programu bez GUI



<http://zofia.kruczkiewicz.staff.iar.pwr.wroc.pl/wyklady/INP002017/Library1SE.rar>

Link do spakowanych dwóch projektów

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

File Edit View Navig Sour Refac Ru Deb Prof Tea Toc Wind Hel 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**

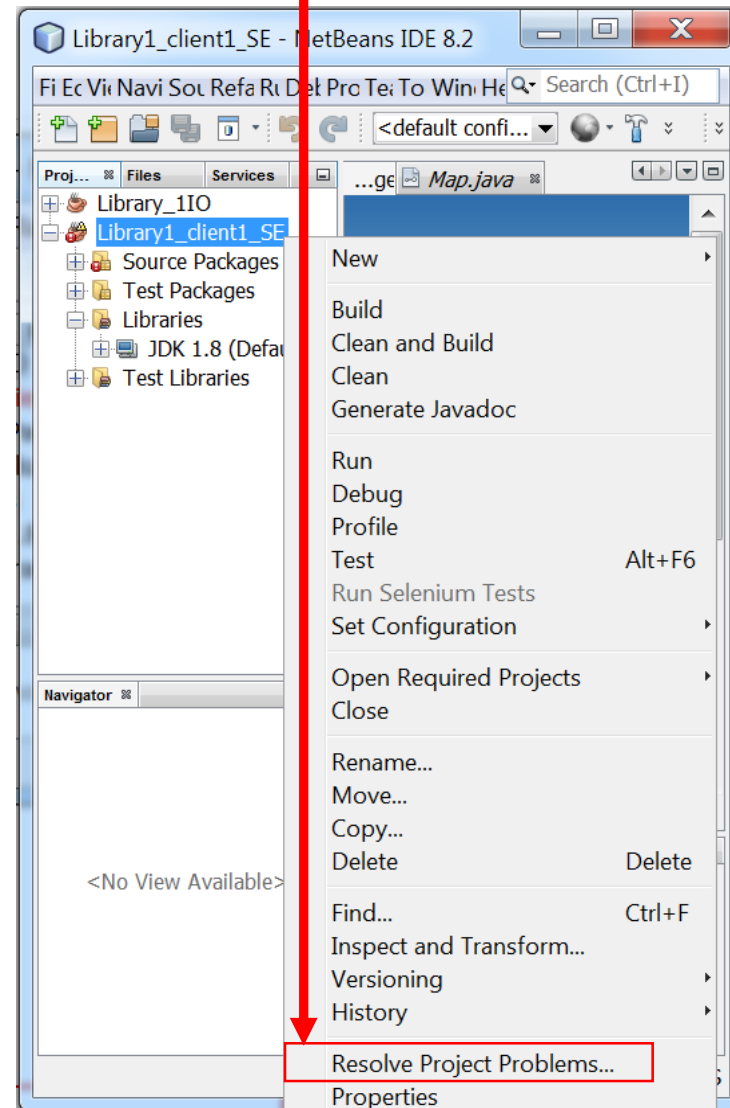
**biznesową** jest

biblioteką dla

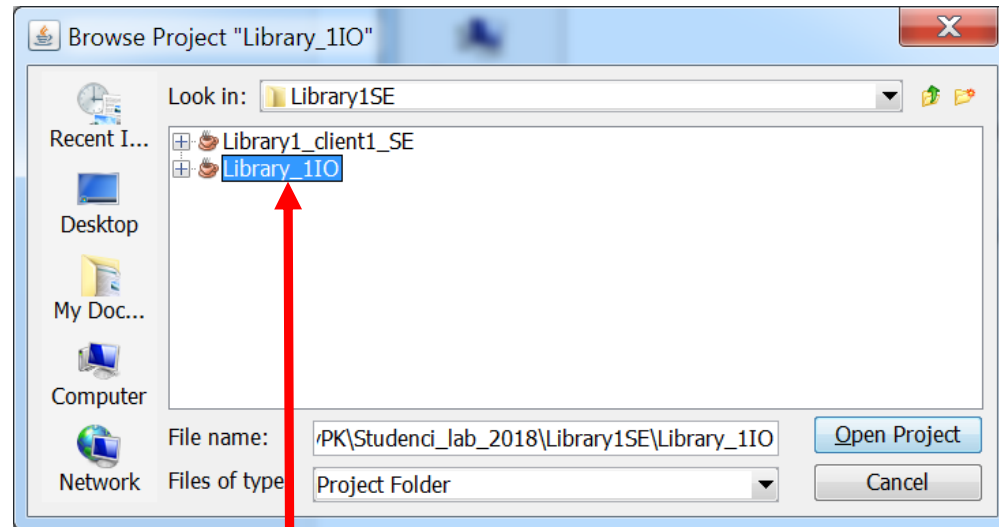
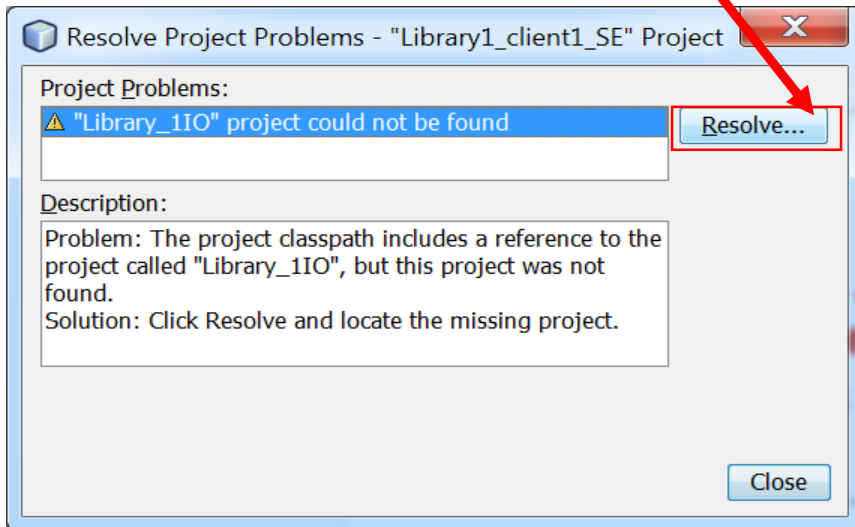
projektu **warstwy**

**klienta**

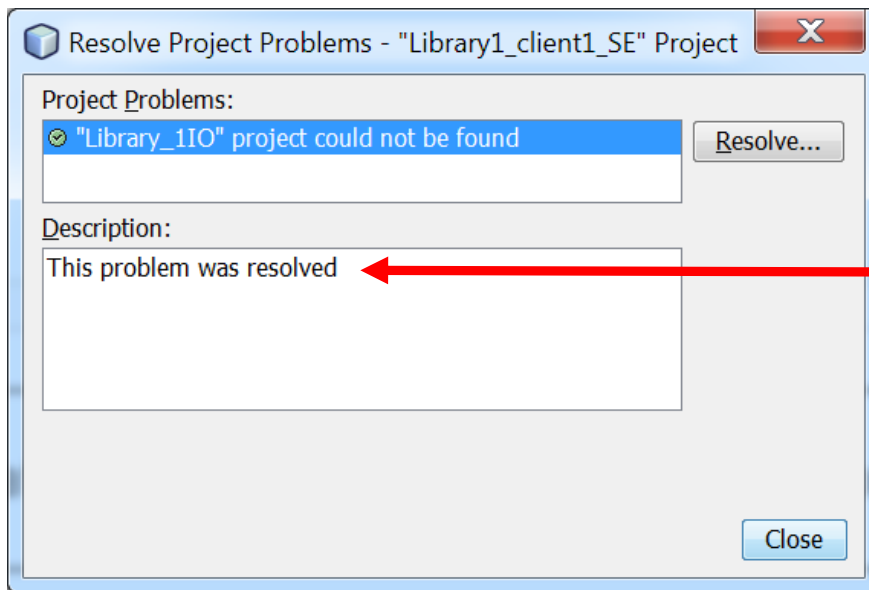
Jeśli po rozpakowaniu dwóch projektów pojawi się informacja o błędzie w projekcie **Library1\_client\_SE**, wtedy należy prawym klawiszem myszy wybrać wybrany ten projekt i wybrać pozycję **Resolve Project Problems...**



Należy nacisnąć klawisz **Resolve...**

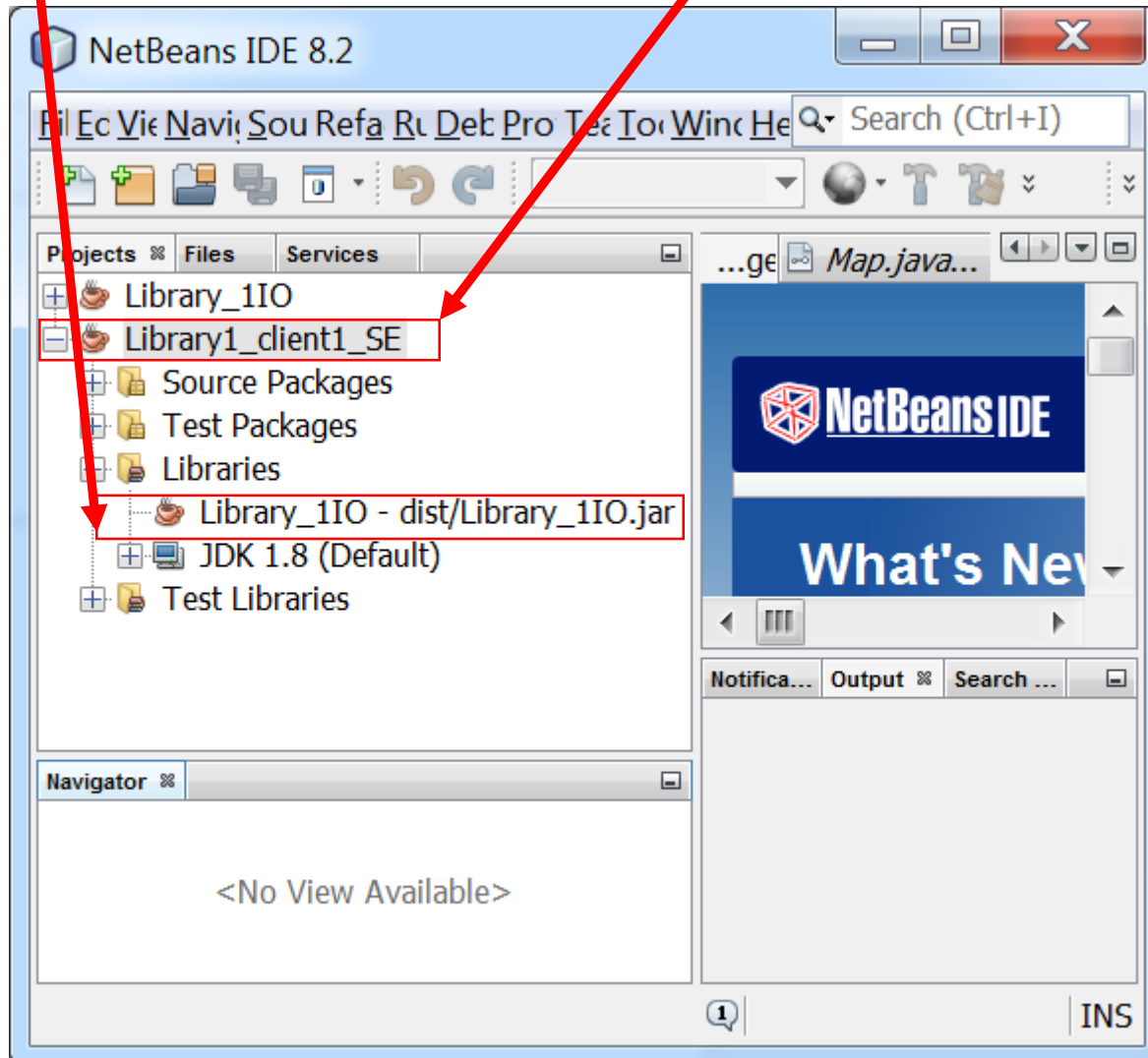


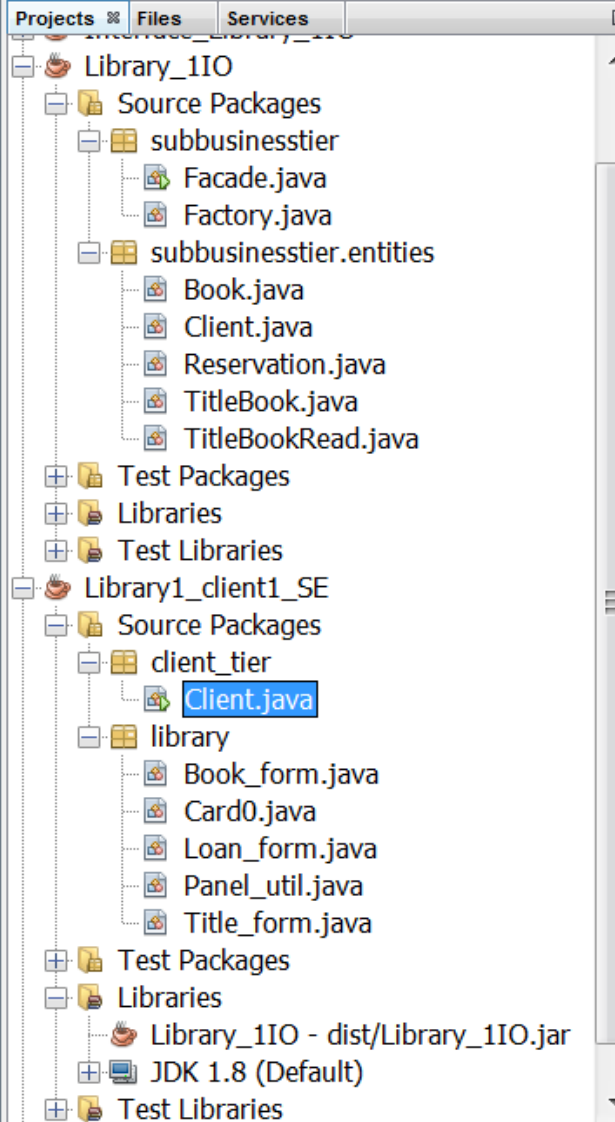
Należy wybrać projekt **Library\_1IO** i nacisnąć klawisz **Open Project**



Problem brak kontaktu z projektem zawierającym logikę biznesową, został rozwiązany

Rezultat – ponownie zostało dodane powiązanie projektu **Library\_1IO** z programem **Library1\_client2\_SE**





```
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 }
```

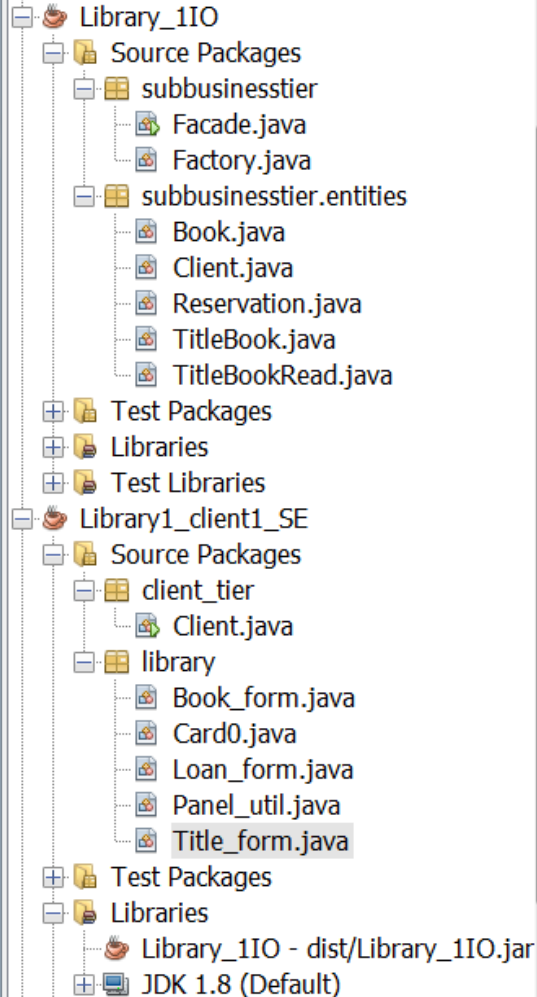
Udostępnianie w programie warstwy klienta logiki biznesowej za pomocą wzorca **Facade**, występującego w roli wzorca **Singleton**

client\_tier.Client &gt; main &gt;

Output

&gt;&gt; Java DB Database Process Library1\_client1\_SE (run)





```
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**

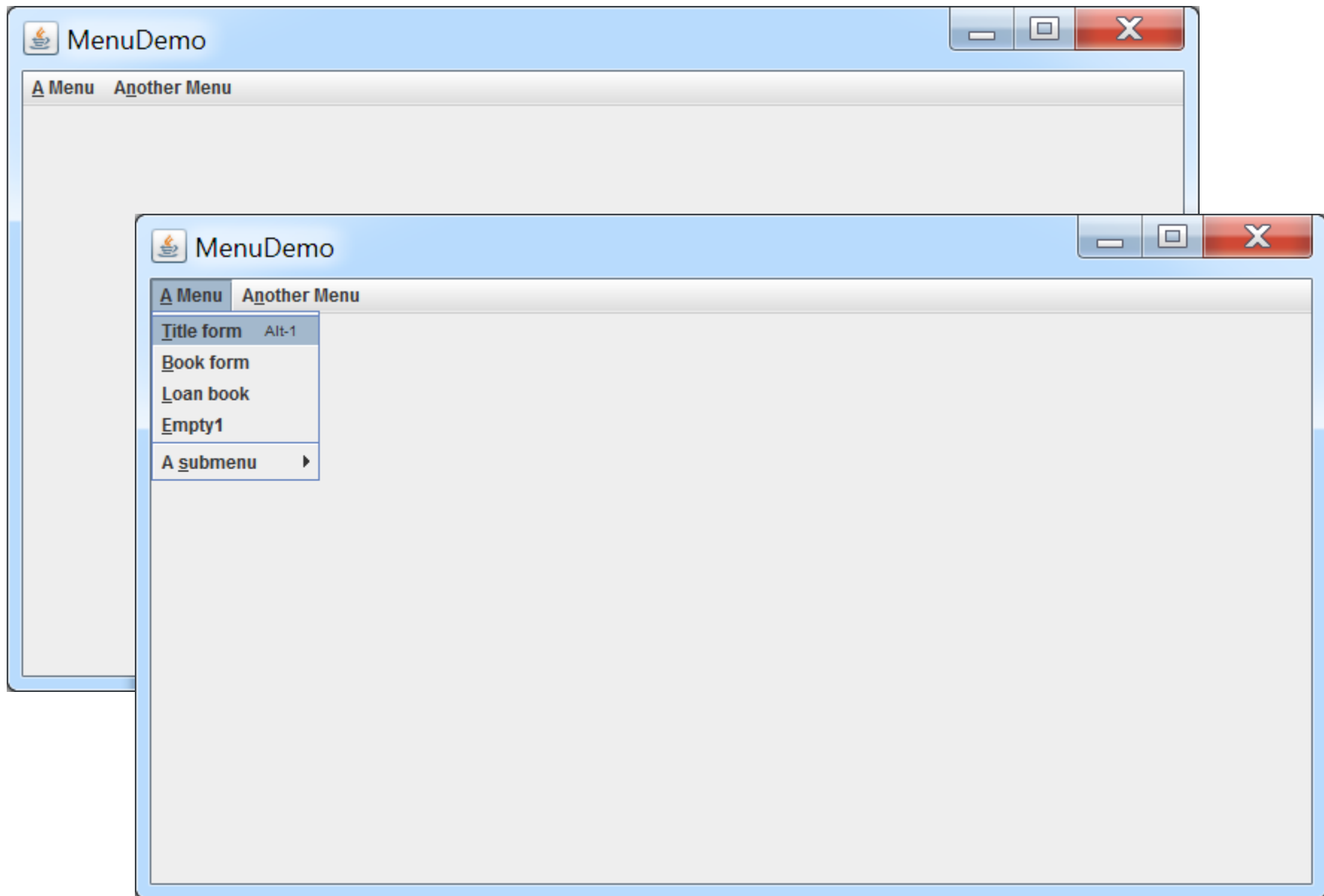
```
void table_content() {  
    Object[][] titles = Client.getFacade().getTitleBooksModel();  
    model.setData(titles);  
}  
  
@Override  
public void actionPerformed(ActionEvent event) {  
    if (!table.getSelectionModel().isSelectionEmpty()) {  
        String what_book_type;  
        if (number.getText().equals("")) {  
            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);  
        }  
    }  
}
```

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);  
    }  
}
```

# Dostęp do formularzy do wprowadzania danych tytułów i książek



# Formularze do wprowadzania 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 ksiązek papierowych i nagranych

**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

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

**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

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

**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

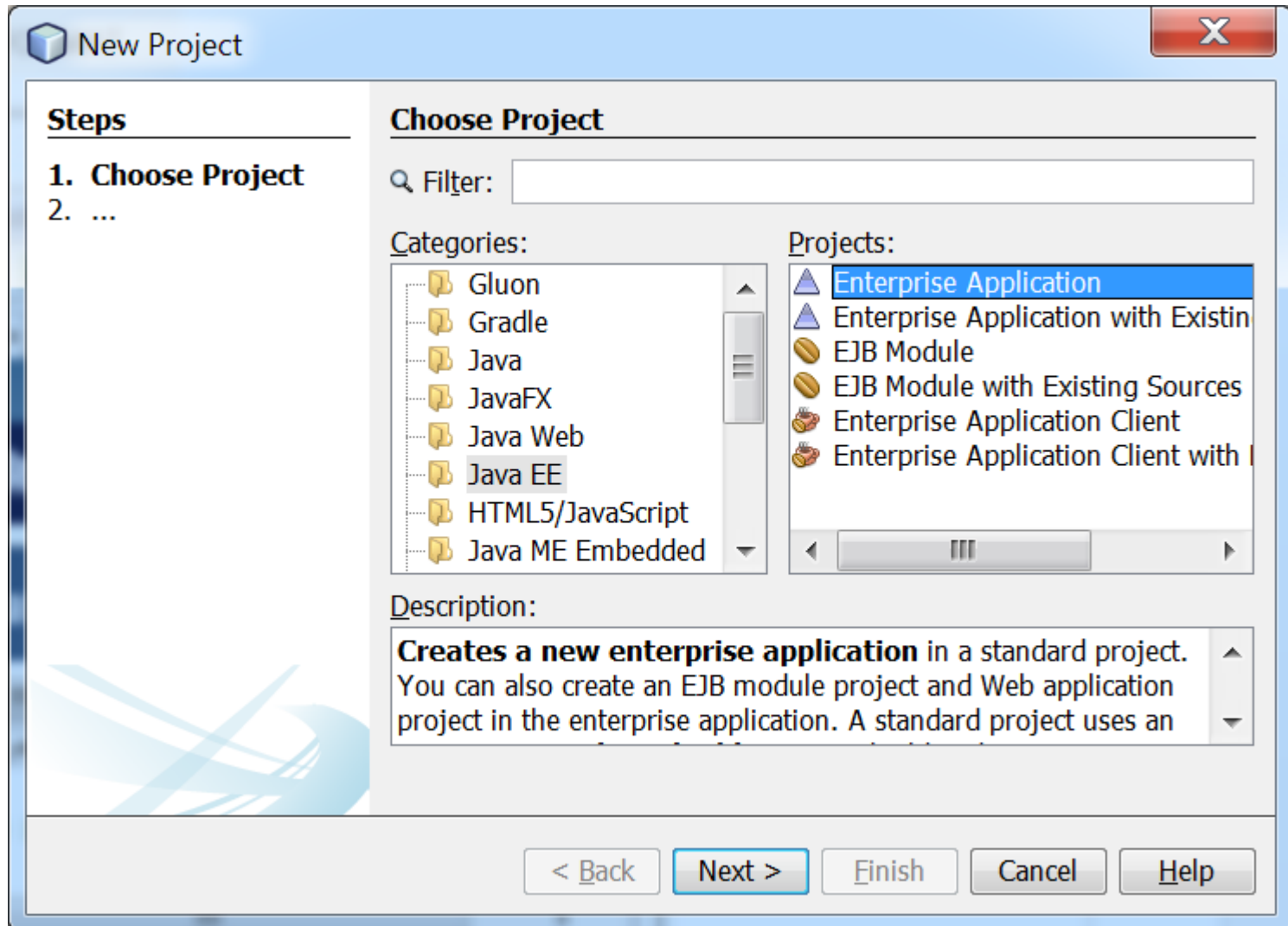
Title: Tytul1 Author: Author1 ISBN: 1234567 Publisher: Publisher1 Actor: Actor1 Number: 1

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

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

**Wykonanie aplikacji typu Enterprise na platformie Java EE7 - wykorzystującej kod wykonany podczas tworzenia aplikacji dwuwarstwowej na platformie SE dla 1 użytkownika**

# 1. Wykonanie projektu głównego: **File/New Project.../JavaEE/Enterprise Application** i naciśnięcie klawisza **Next**



## 2. Podanie nazwy projektu (**Project Name**) oraz wybór lokalizacji (**Project Location**) i naciśnięcie klawisza **Next**

The screenshot shows the 'New Enterprise Application' wizard at the 'Name and Location' step. The 'Steps' list on the left indicates the current step is '2. Name and Location'. The main area contains the following fields and options:

- Project Name:** Library1\_EE
- Project Location:** C:\Studia\Library2 (with a 'Browse...' button)
- Project Folder:** C:\Studia\Library2\Library1\_EE
- Use Dedicated Folder for Storing Libraries
- Libraries Folder:** (empty) (with a 'Browse...' button)

At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Next >' button is highlighted in blue.

**Uwaga: w nazwach katalogów należy stosować jedynie znaki z alfabetu angielskiego (duże i małe litery)**

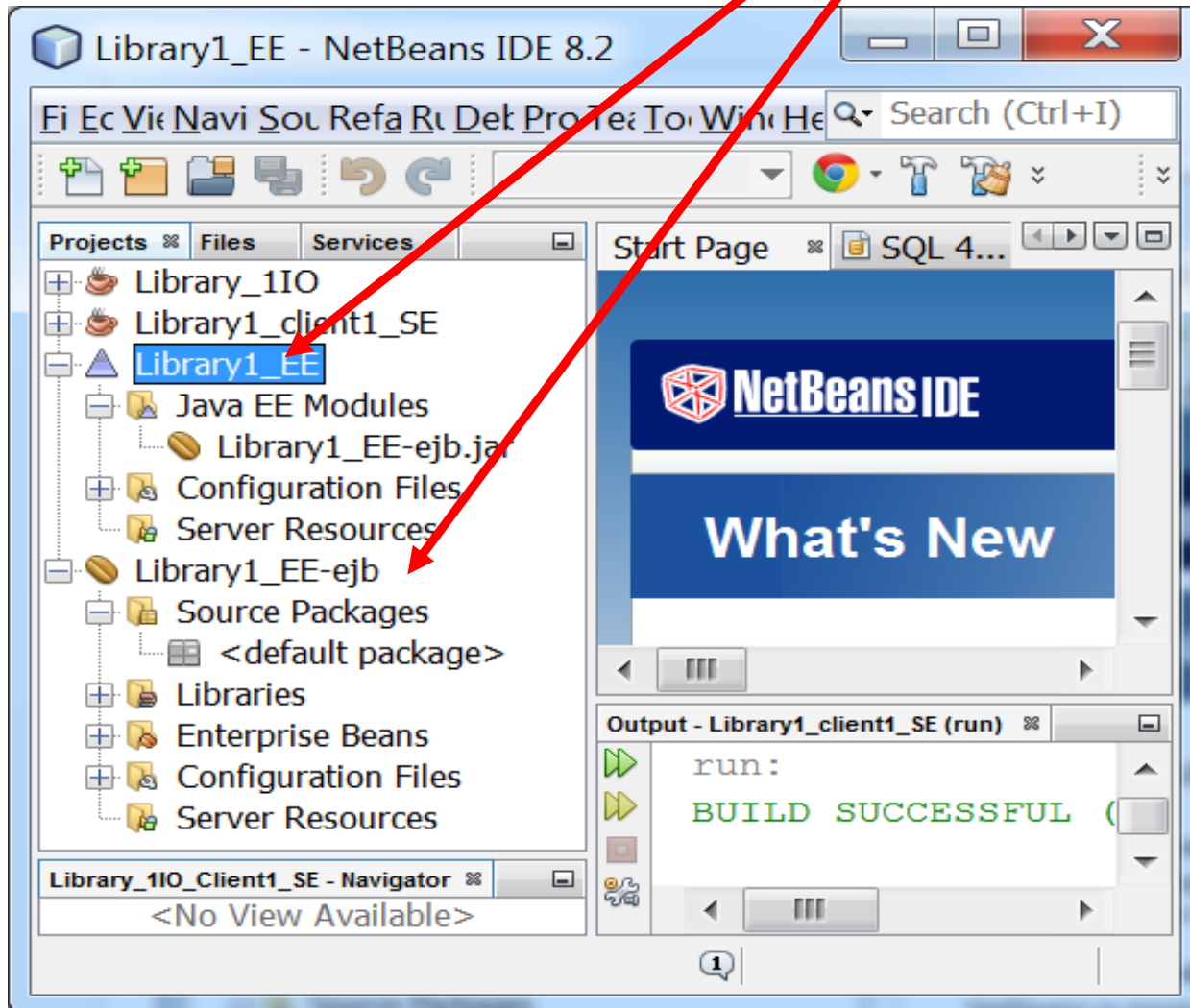
## 3. Wybór serwera aplikacji (**Server**) oraz wybór utworzenia tylko modułu EJB (**Create EJB Module**) i naciśnięcie **Finish**

The screenshot shows the 'New Enterprise Application' wizard at the 'Server and Settings' step. The 'Steps' list on the left indicates the current step is '3. Server and Settings'. The main area contains the following fields and options:

- Server:** GlassFish Server (with an 'Add...' button)
- Java EE Version:** Java EE 7
- Create EJB Module: Library1\_EE-ejb
- Create Web Application Module: Library1\_EE-war

At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'. The 'Finish' button is highlighted in blue.

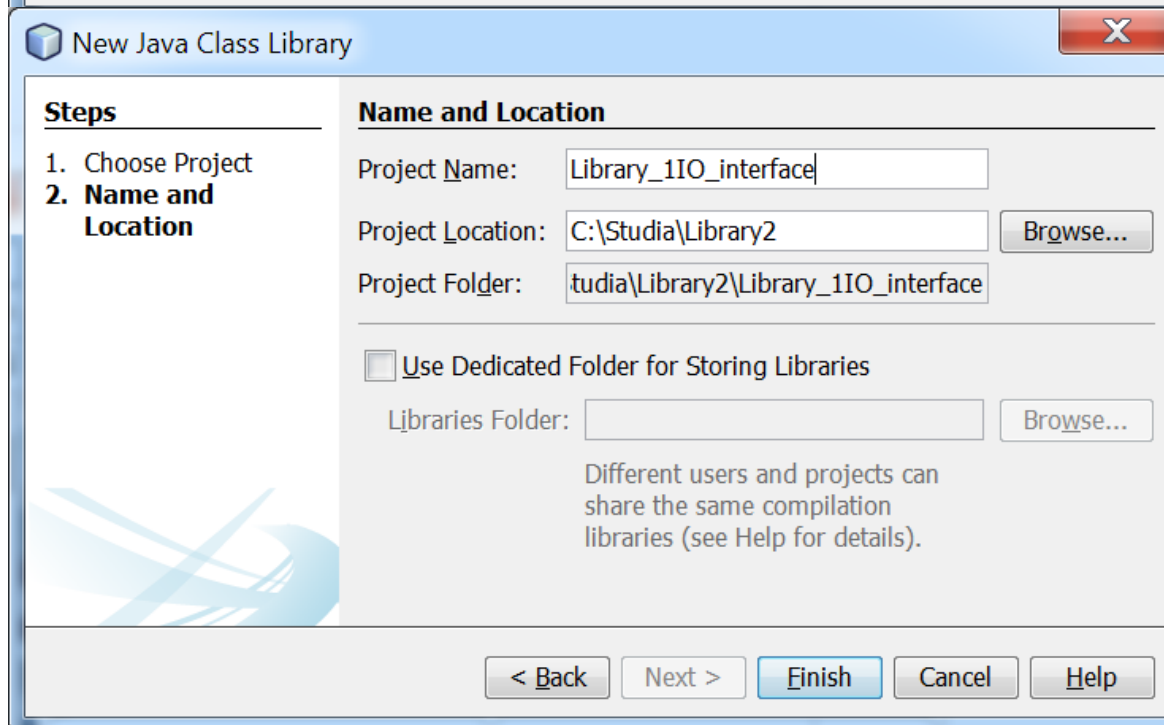
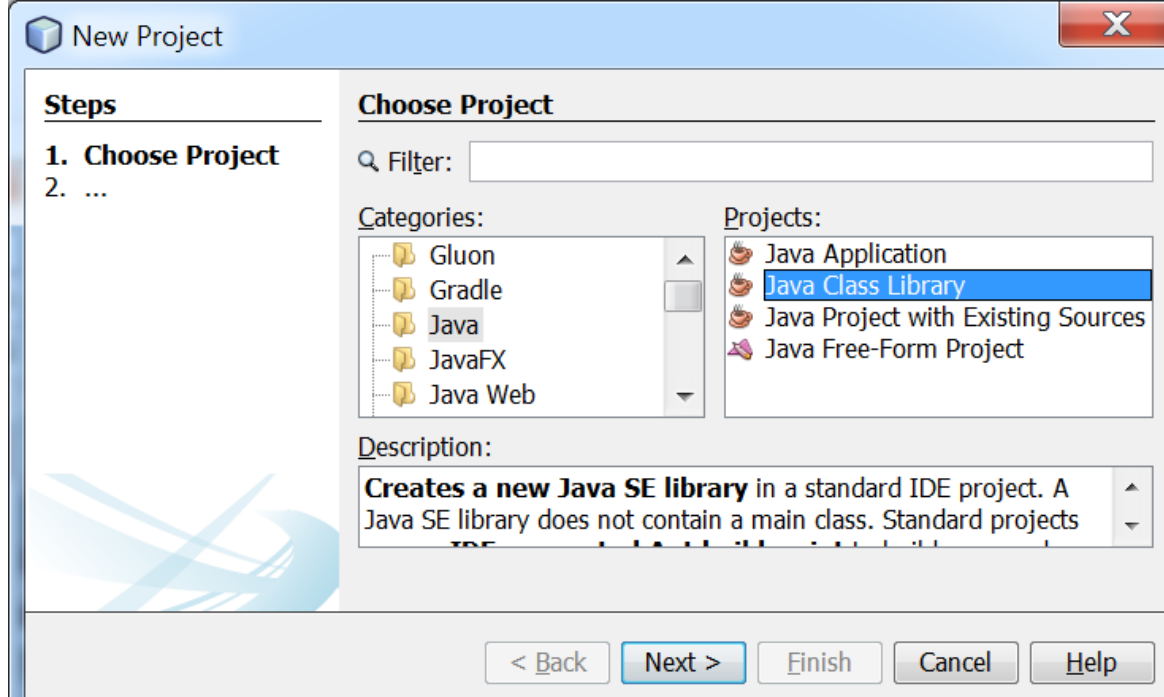
#### 4. Wynik: założony został projekt główny typu Enterprise Application oraz pusty projekt typu moduł EJB



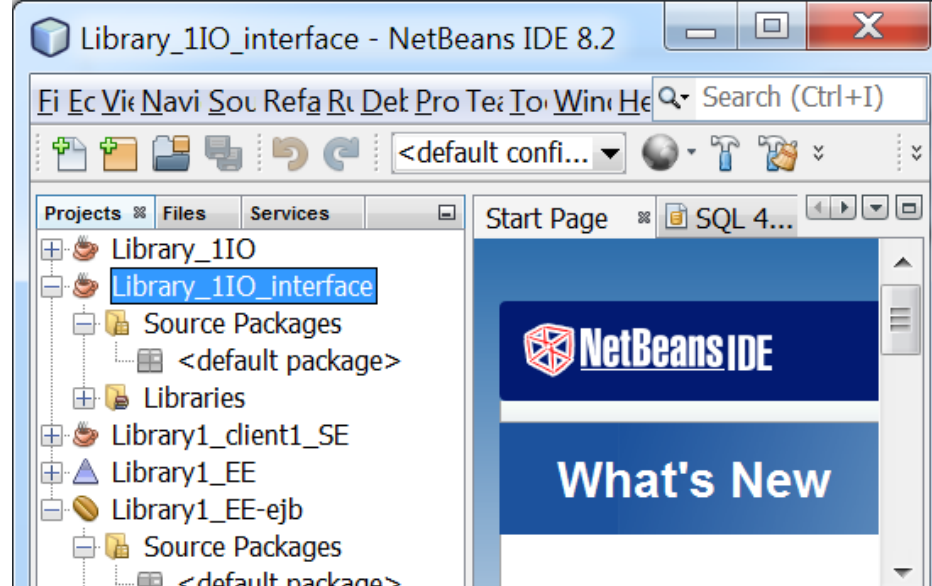


5. Wykonanie projektu typu Java ClassLibrary do przechowania interejosu komponentu EJB typu Session Bean do uzdalnienia dostępu do metod logiki boznesowej klasy Facade

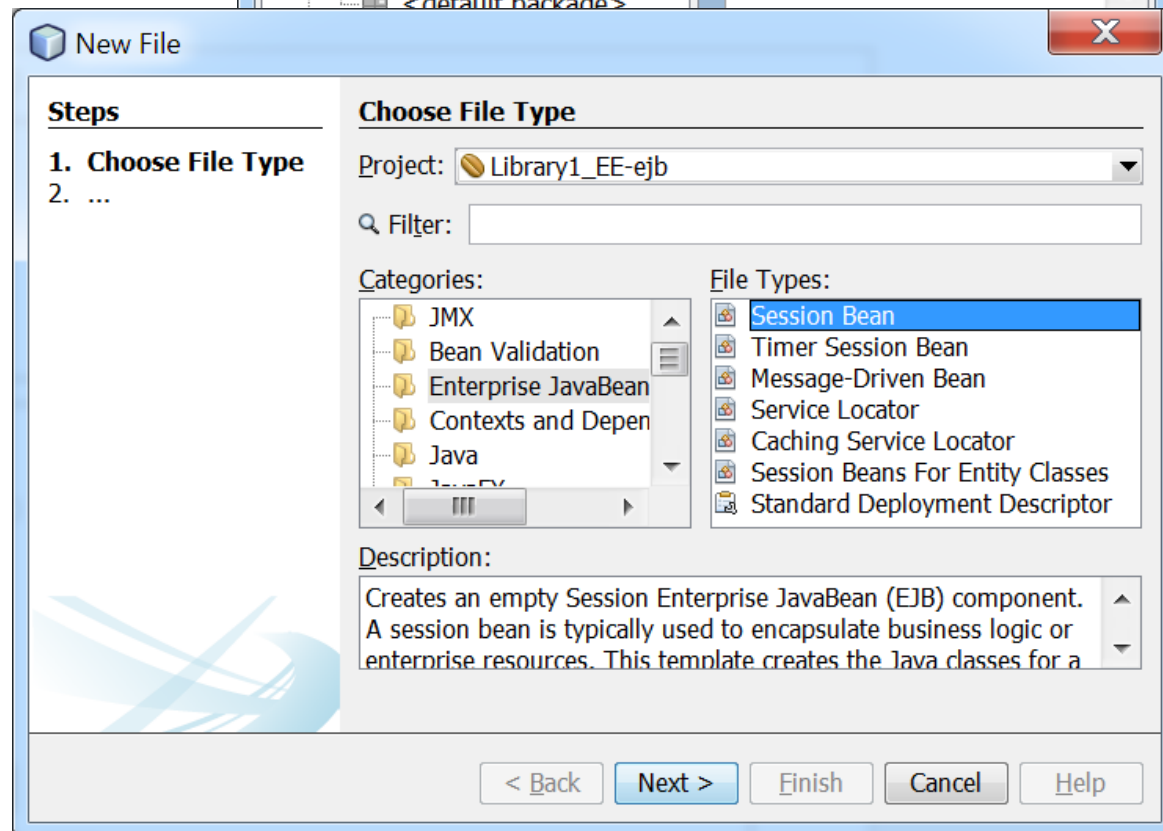
6. Należy wybrać: **File/New Project.../Java/Java Class Library** i nadać mu nazwę (**Project Name**) oraz lokalizację (**Project Location**) – taką samą, jak projekt typu Enterprise Application



**7. Wykonany projekt** typu Java ClassLibrary do przechowania interejosu komponentu EJB typu Session Bean do uzdalnienia dostępu do metod logiki biznesowej klasy Facade



**8. Wstawienie komponentu EJB typu Session Bean do modułu EJB do uzdalnienia dostępu do metod logiki biznesowej klasy Facade:**  
prawym klawiszem myszy wybrać projekt typu moduł EJB, i wybrać pozycje:  
**New/Other.../Enterprise JavaBean/Session Bean** i nacisnąć na klawisz **Next**



9. W polu **EJB Name** wpisać nazwę komponentu, w polu **Package** wpisać nazwę pakietu, w którym będzie przechowany tworzony obecnie komponent EJB. Należy wybrać w **Session Type** pozycję **Stateless** i w polu **Create Interface** zaznaczyć **Remote**. Wtedy z listy należy wybrać utworzony wcześniej projekt typu Java Class Library. Następnie należy nacisnąć klawisz **Finish**.

**New Session Bean**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

EJB Name: EJBFacade

Project: Library1\_EE-ejb

Location: Source Packages

Package: businessstier

Session Type:

Stateless

Stateful

Singleton

Create Interface:

Local

Remote in project: Library\_1IO\_interface

< Back   Next >   **Finish**   Cancel   Help

**10. W projekcie typu Java Class Library utworzył się pusty interfejs o nazwie komponentu EJB typu Session Bean z przyrostkiem Remote. W projekcie typu moduł EJB utworzył się pusty komponent typu EJB implementujący dany interfejs – też pusty.**

```
Library_1IO_interface - NetBeans IDE 8.2
File Edit View Navigat Source Refacto Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Projects Files Services ...17 EJBFacade.java EJBFacadeRemote.java
Source History
1 package businessstier;
2
3
4 import javax.ejb.Remote;
5
6
7 @Remote
8 public interface EJBFacadeRemote {
9
10 }
```

```
Library1_EE-ejb - NetBeans IDE 8.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help Search (Ctrl+I)
Projects Files Services ...16 SQL 17 EJBFacade.java EJBFacadeRemote.java
Source History
1 package businessstier;
2
3 import javax.ejb.Stateless;
4
5 @Stateless
6 public class EJBFacade implements EJBFacadeRemote {
7
8
9 }
10
businessstier.EJBFacade >
Output - Library1_client1_SE (run)
run:
BUILD SUCCESSFUL (total time: 8 seconds)
```

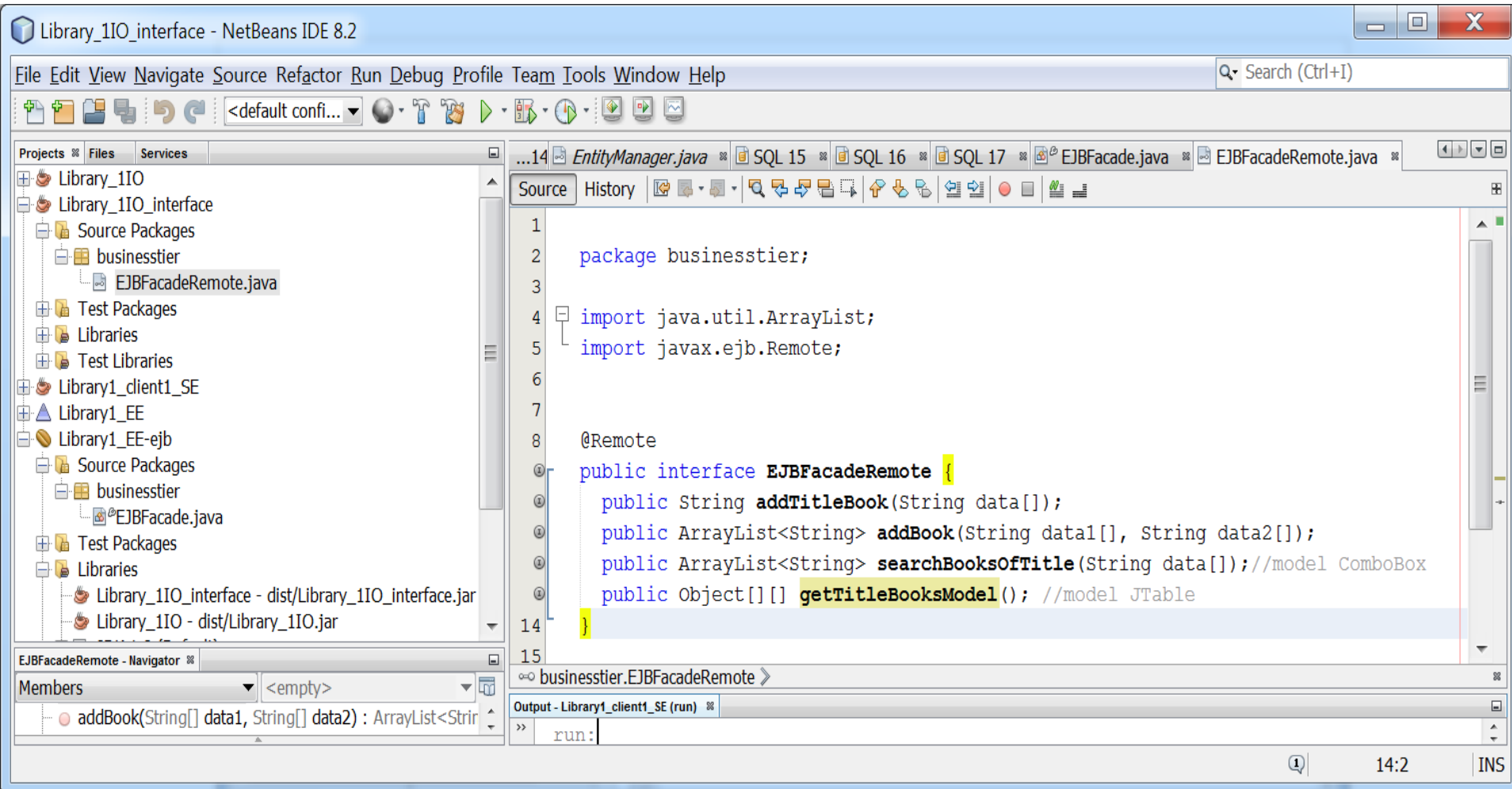
11. Należy w projekcie modułu EJB w folderze Libraries dodać **projekt z logiką biznesową**. Należy zdefiniować obiekt typu **Facade** i zdefiniować te metody, które były wywołane w projekcie warstwy klienta. Ciało tych metod jest oparte na metodach klasy Facade.

The screenshot displays an IDE interface with the following components:

- Project Explorer (Left):** Shows a project structure with a 'Libraries' folder containing 'Library\_1IO\_interface.jar' and 'Library\_1IO - dist/Library\_1IO.jar'. A red arrow points from the text above to the 'Library\_1IO\_interface.jar' entry.
- Source Editor (Center):** Displays the code for 'EJBFacade.java'. The code defines a stateless EJB class that implements 'EJBFacadeRemote'. It includes imports for 'java.util.ArrayList', 'javax.ejb.Stateless', and 'subbusinesstier.Facade'. The class contains several methods: a constructor, 'addTitleBook', 'addBook', 'searchBooksOfTitle', and 'getTitleBooksModel'. A red arrow points from the text above to the 'Facade facade = new Facade();' line in the constructor.
- Members (Bottom Left):** Shows the members of the 'EJBFacade' class, including 'addBook', 'addTitleBook', 'getTitleBooksModel', 'searchBooksOfTitle', and 'facade'.
- Output (Bottom):** Shows the output of a run, which is 'run:'.

```
1 package businesstier;
2
3 import java.util.ArrayList;
4 import javax.ejb.Stateless;
5 import subbusinesstier.Facade;
6
7 @Stateless
8 public class EJBFacade implements EJBFacadeRemote {
9
10     Facade facade = new Facade();
11
12     @Override
13     public String addTitleBook(String data[]) {
14         return facade.addTitleBook(data);
15     }
16
17     @Override
18     public ArrayList<String> addBook(String data[], String data2[]) {
19         return facade.addBook(data1, data2);
20     }
21
22     @Override
23     public ArrayList<String> searchBooksOfTitle(String data[]) { //model ComboBox
24         return facade.searchBooksOfTitle(data);
25     }
26
27     @Override
28     public Object[][] getTitleBooksModel() { //model jTable
29         return facade.getTitleBooksModel();
30     }
31 }
```

## 12. Należy w projekcie typu Java Class Library w interfejsie komponentu EJB zadeklarować metody zdefiniowane w komponencie EJB, znajdującym się w projekcie modułu EJB



The screenshot displays the NetBeans IDE 8.2 interface. The main editor window shows the source code for the `EJBFacadeRemote` interface. The code is as follows:

```
1  
2 package busnesstier;  
3  
4 import java.util.ArrayList;  
5 import javax.ejb.Remote;  
6  
7  
8 @Remote  
9 public interface EJBFacadeRemote {  
10     public String addTitleBook(String data[]);  
11     public ArrayList<String> addBook(String data1[], String data2[]);  
12     public ArrayList<String> searchBooksOfTitle(String data[]); //model ComboBox  
13     public Object[][] getTitleBooksModel(); //model JTable  
14 }  
15
```

The interface is annotated with `@Remote`. The methods are `addTitleBook`, `addBook`, `searchBooksOfTitle`, and `getTitleBooksModel`. The `searchBooksOfTitle` method is commented with `//model ComboBox` and the `getTitleBooksModel` method is commented with `//model JTable`.

The left sidebar shows the project structure for `Library_1IO_interface`, including `Source Packages` and `Test Packages`. The `businessstier` package contains `EJBFacadeRemote.java`. The bottom status bar shows the time as 14:2 and the keyboard layout as INS.

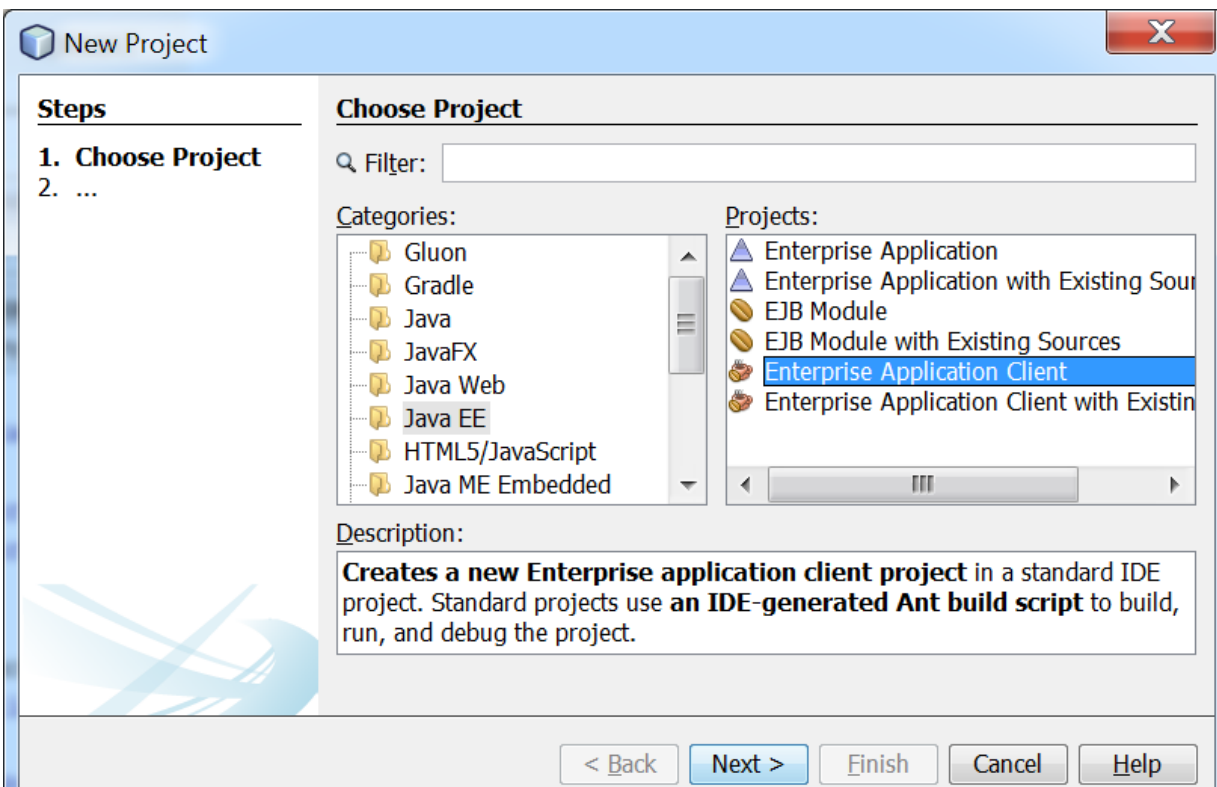


# 13. Rezultat – obecnie adnotacje Override są poprawne w komponencie EJB.

The screenshot shows an IDE window with the following components:

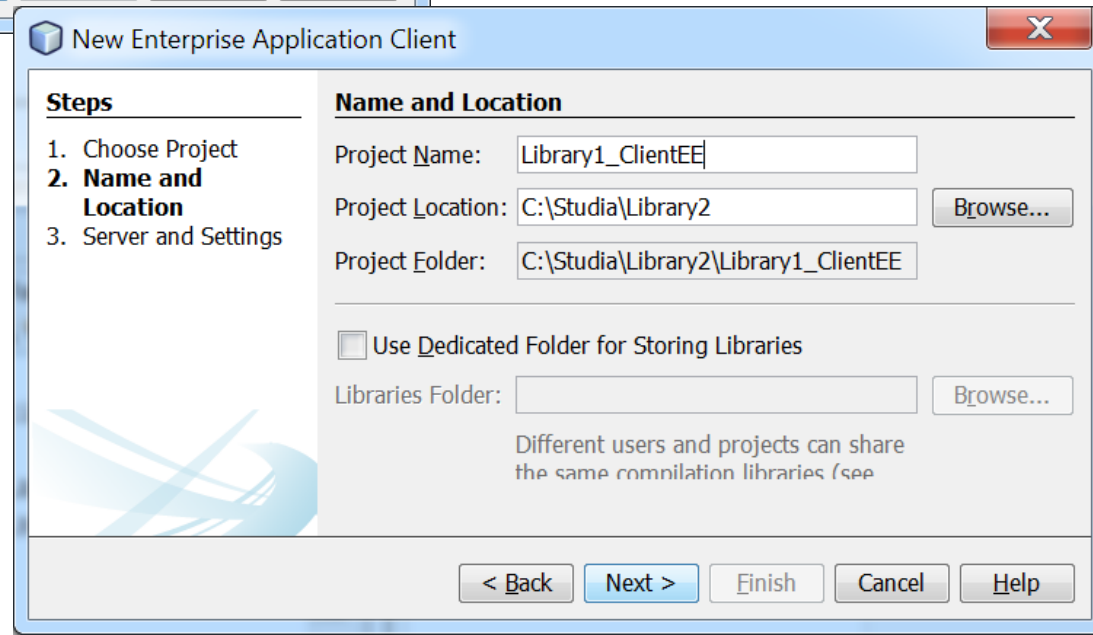
- Menu Bar:** File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. Search (Ctrl+I) is visible on the right.
- Project Explorer (Left):** Shows a project structure with folders like Library\_110, Library\_110\_interface, Library1\_client1\_SE, and Library1\_ClientEE. The Library1\_ClientEE folder is expanded, showing Source Packages (client\_tier, library) and Test Packages (Libraries, Test Libraries, Configuration Files, Server Resources).
- Source Editor (Center):** Displays the code for `businessstier.EJBFacade`. The code includes imports for `javax.ejb.Stateless` and `subbusinessstier.Facade`. The `EJBFacade` class implements `EJBFacadeRemote` and contains several methods, each annotated with `@Override`:

```
4 import javax.ejb.Stateless;
5 import subbusinessstier.Facade;
6
7 @Stateless
8 public class EJBFacade implements EJBFacadeRemote {
9
10     Facade facade = new Facade();
11
12     @Override
13     public String addTitleBook(String data[]) {
14         return facade.addTitleBook(data);
15     }
16     @Override
17     public ArrayList<String> addBook(String data1[], String data2[]) {
18         return facade.addBook(data1, data2);
19     }
20     @Override
21     public ArrayList<String> searchBooksOfTitle(String data[]) { //model ComboBox
22         return facade.searchBooksOfTitle(data);
23     }
24     @Override
25     public Object[][] getTitleBooksModel() { //model jTable
26         return facade.getTitleBooksModel();
27     }
28 }
```
- Members Navigator (Bottom Left):** Shows the members of the `EJBFacade` class, including `addBook`, `addTitleBook`, `getTitleBooksModel`, `searchBooksOfTitle`, and `facade`.
- Output Window (Bottom):** Shows the status of the application, including `Java DB Database Process`, `GlassFish Server`, and `Library1_ClientEE (run) #2`.



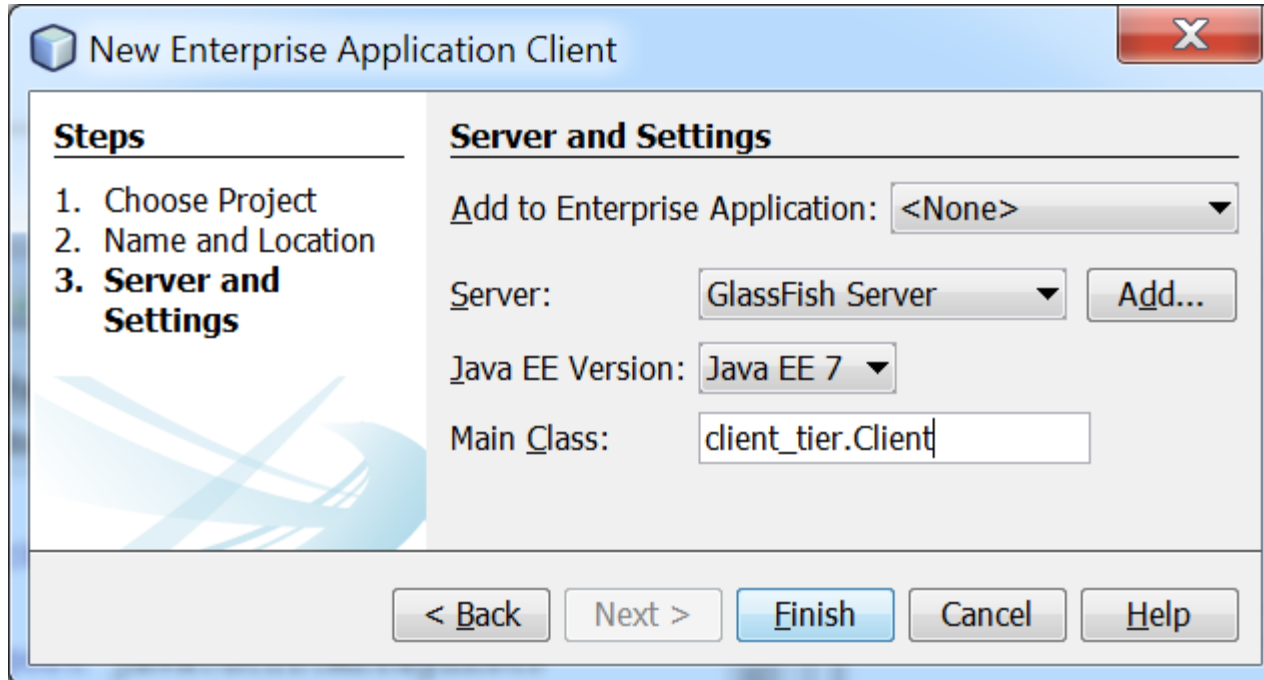
**14. Należy wykonać projekt typu Enterprise Application Client: File/New Project.../JavaEE/Enterprise Application Client i nacisnąć klawisz Next**

**15. Należy wypełnić pole Project Name podając nazwę projektu oraz Project Location podając nazwę katalogu projektu i należy nacisnąć klawisz Next**

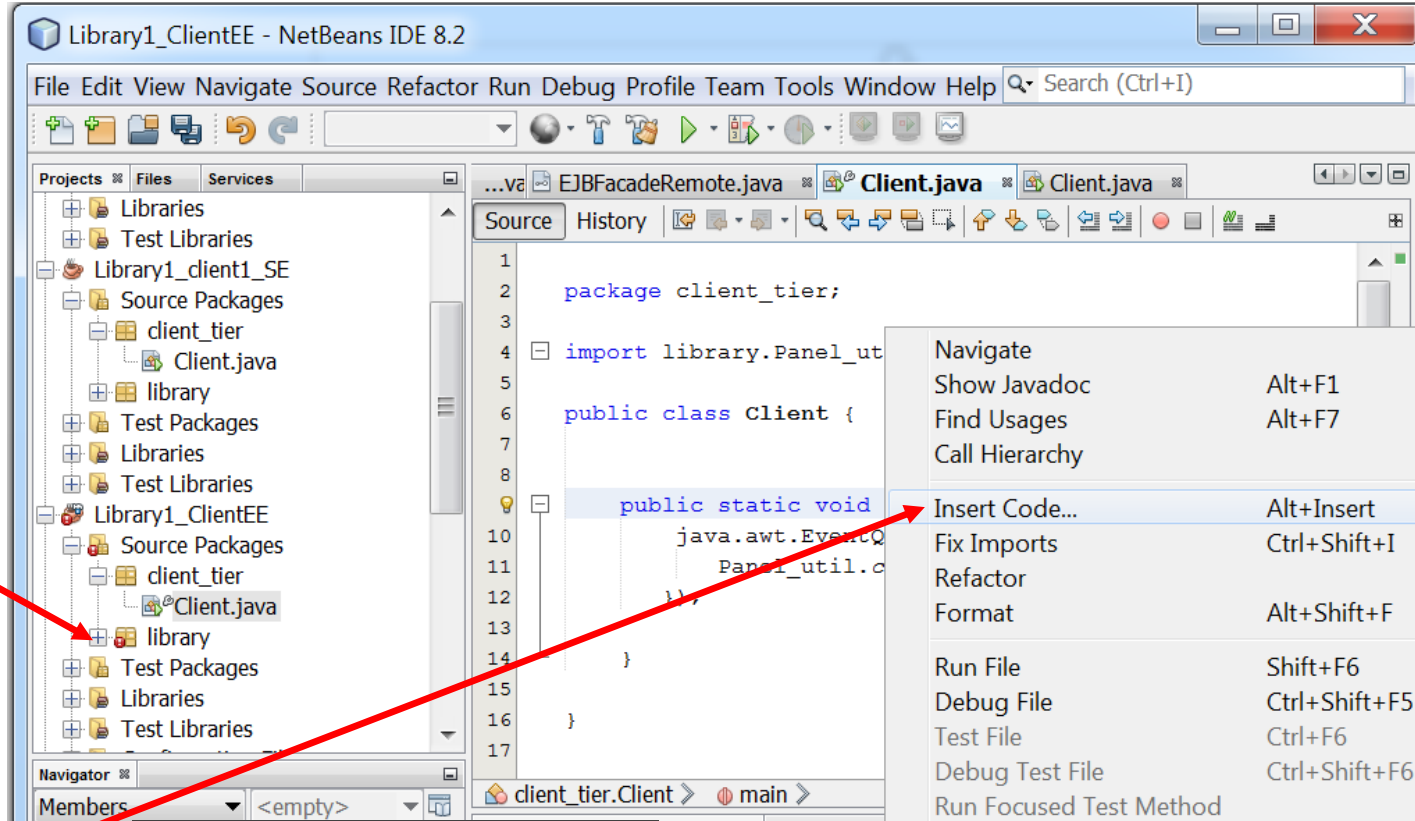




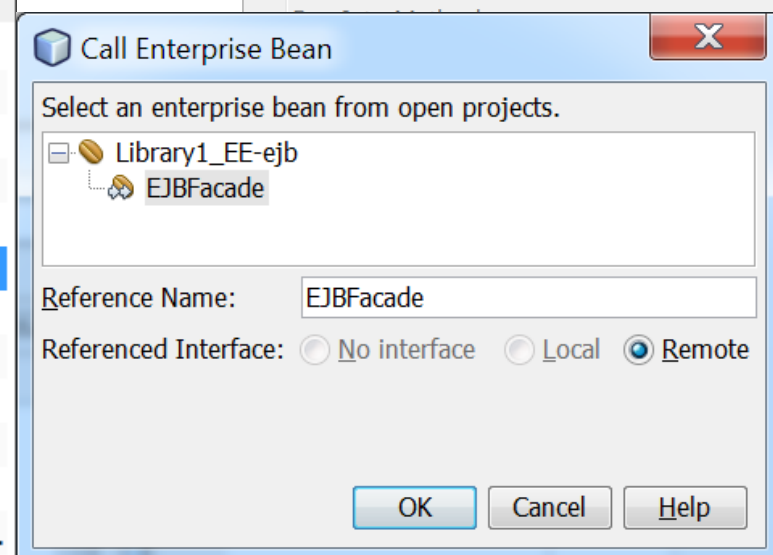
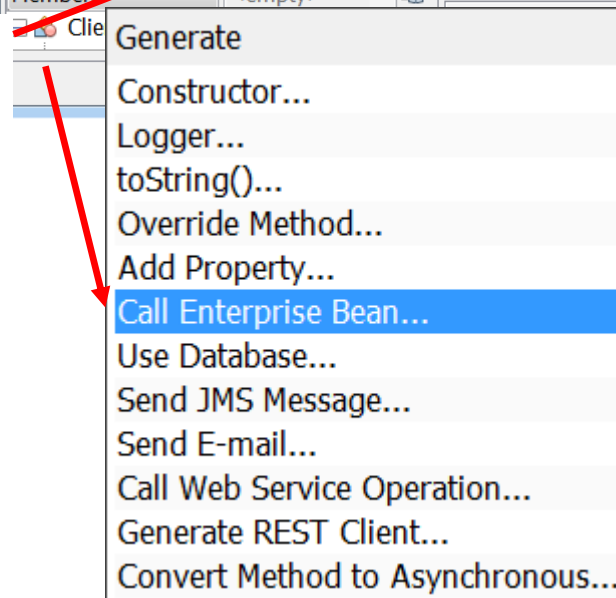
16. Kod tworzonego projektu typu Enterprise Application Client będzie wykonany w opaci o pakiet **library** z projektu warstwy klienta (aplikacja dwuwarstwowa). Zdefiniowano domyślną klasę **Client** w pakiecie **client\_tier** (pole **Main Class**) w celu uniknięcia modyfikacji kodu klas w pakiecie **library**.



17. Skopiowanie pakietu **library** z aplikacji reprezentującej aplikację warstwy klienta.



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



## 19. 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 }
```

main - Navigator

Members

<empty>

main(String[] args)

client\_tier.Client > main >

Output - Library1\_client1\_SE (run)

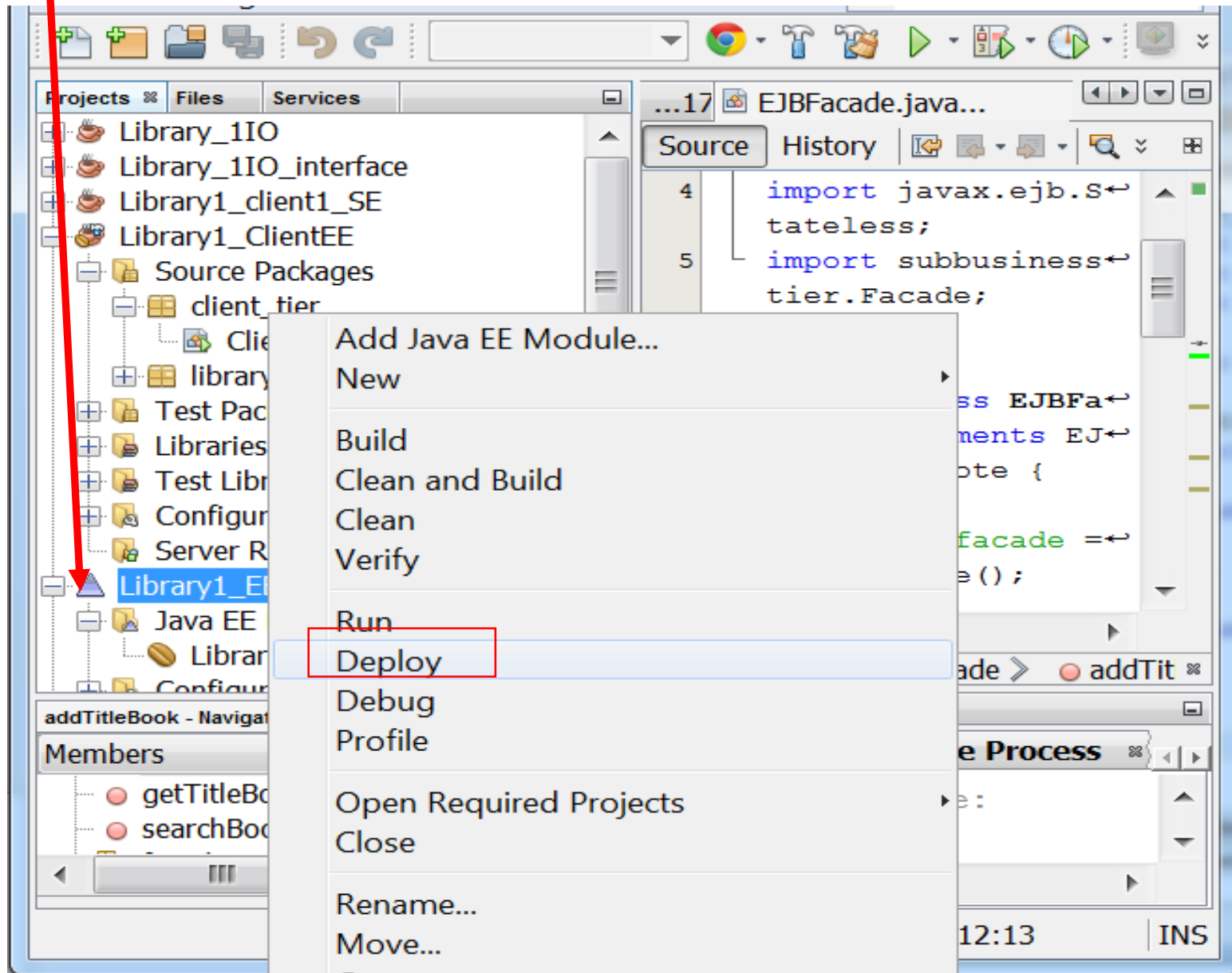
>> run:



21:6

INS

## 20. Uruchomienie aplikacji typu Enterprise Application za pomocą Deploy



## 21. Uruchomiona aplikacja wielowarstwowa typu Enterprise Application

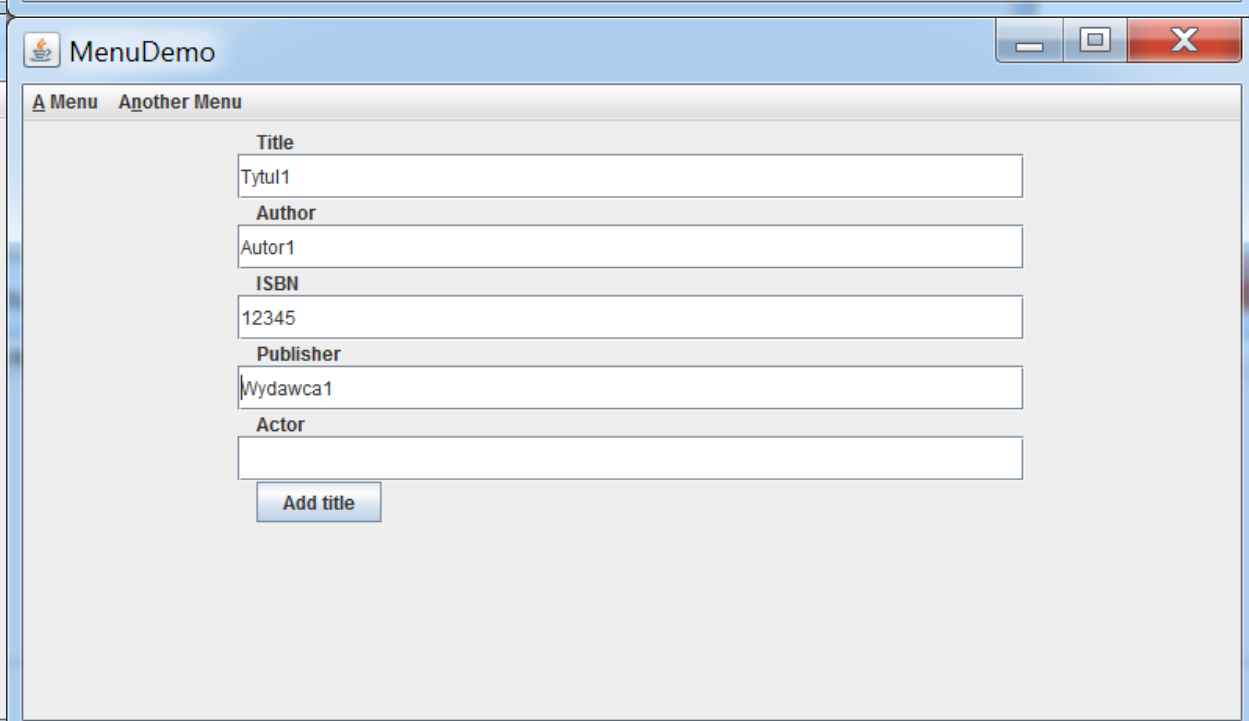
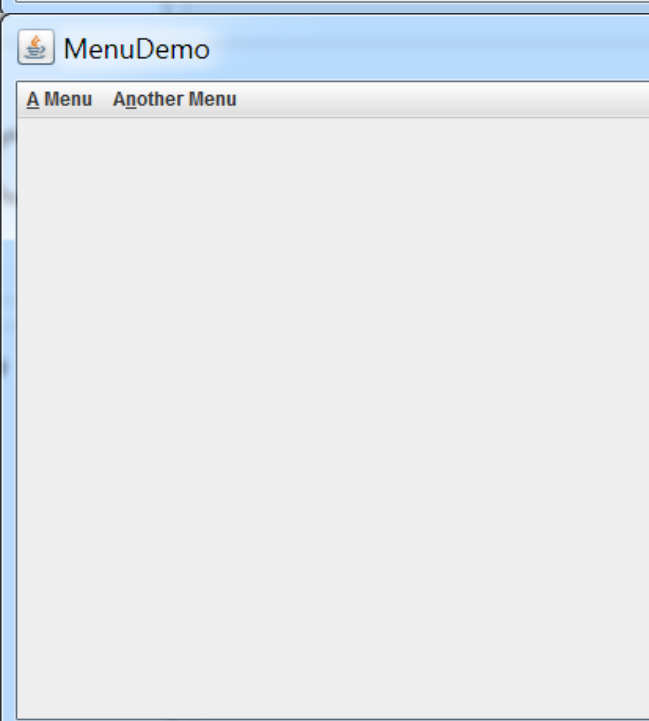
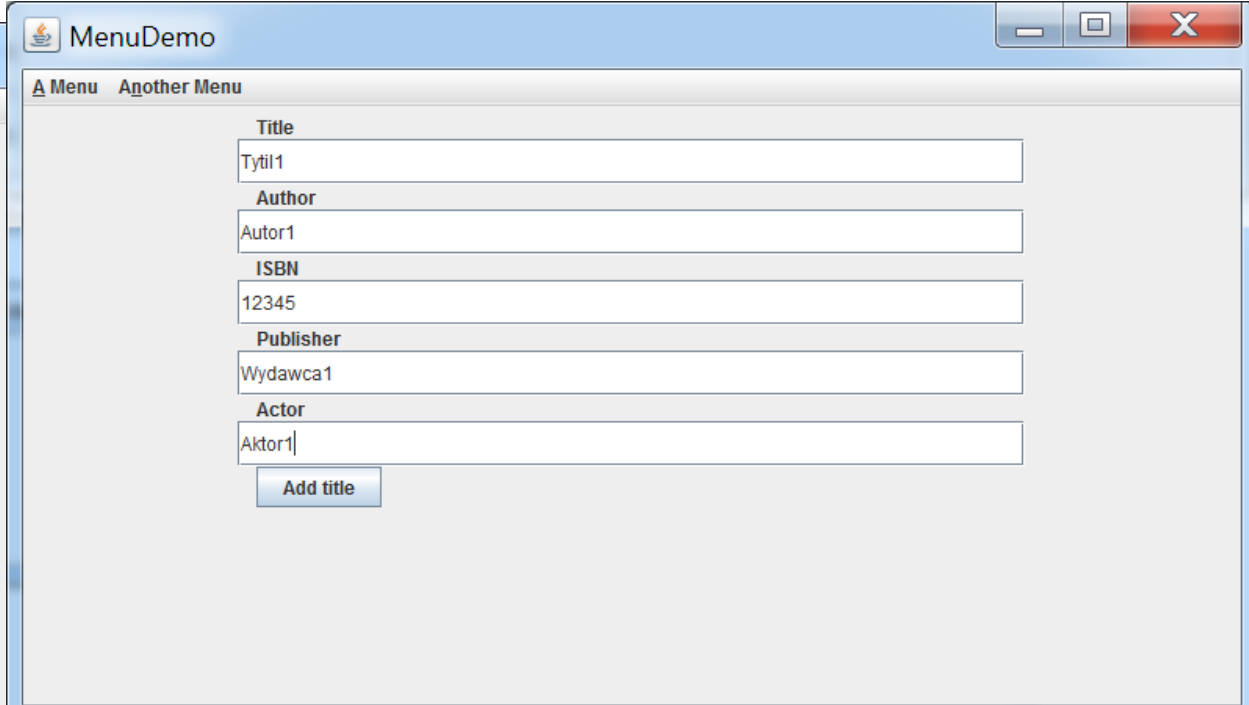
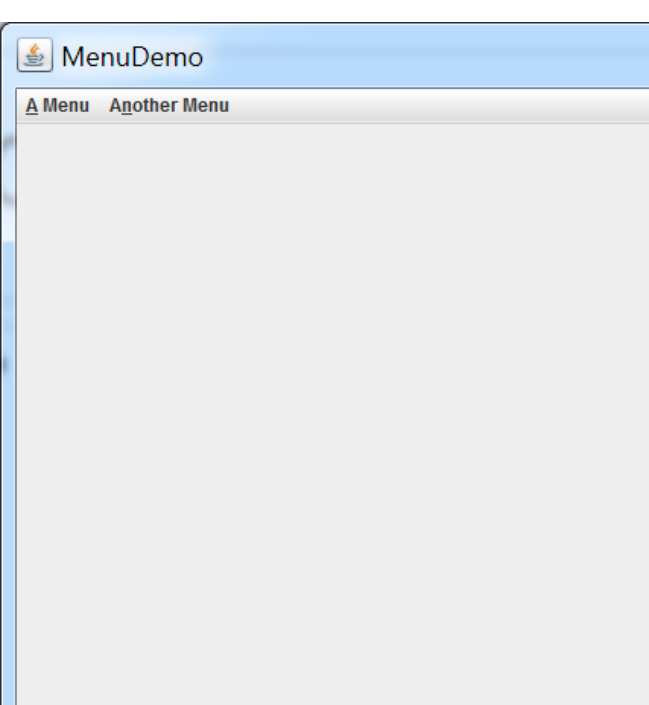
The screenshot displays the NetBeans IDE 8.2 environment. The title bar reads 'Library1\_EE-ejb - NetBeans IDE 8.2'. The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, and Help. The toolbar contains various icons for file operations and development actions. The project explorer on the left shows a tree structure for 'Library1\_EE-ejb', with 'Library1\_EE-ejb.jar' selected. A red arrow points from this selection to the console window. The source editor shows the following code:

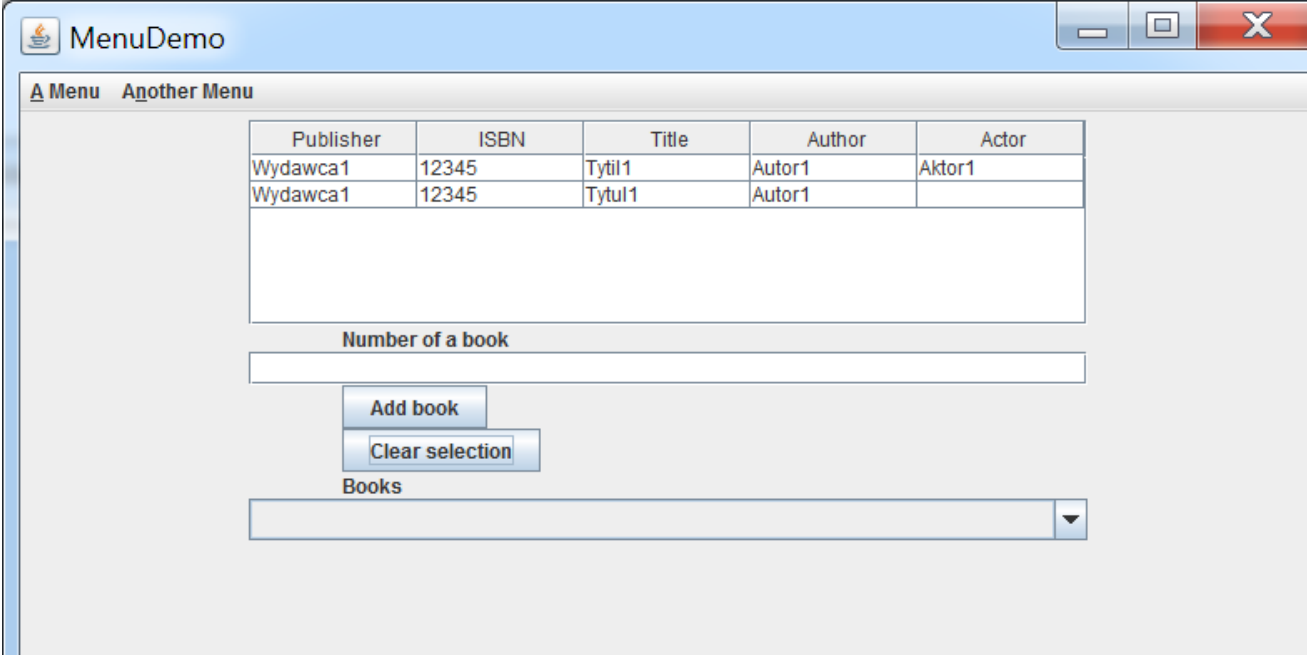
```
1 package businesstier;
2 import java.util.ArrayList;
3 import javax.ejb.Stateless;
4 import subbusinesstier.Facade;
5
6 @Stateless
7 public class EJBFacade implements EJBFacadeRemote {
```

The console window at the bottom shows the following output:

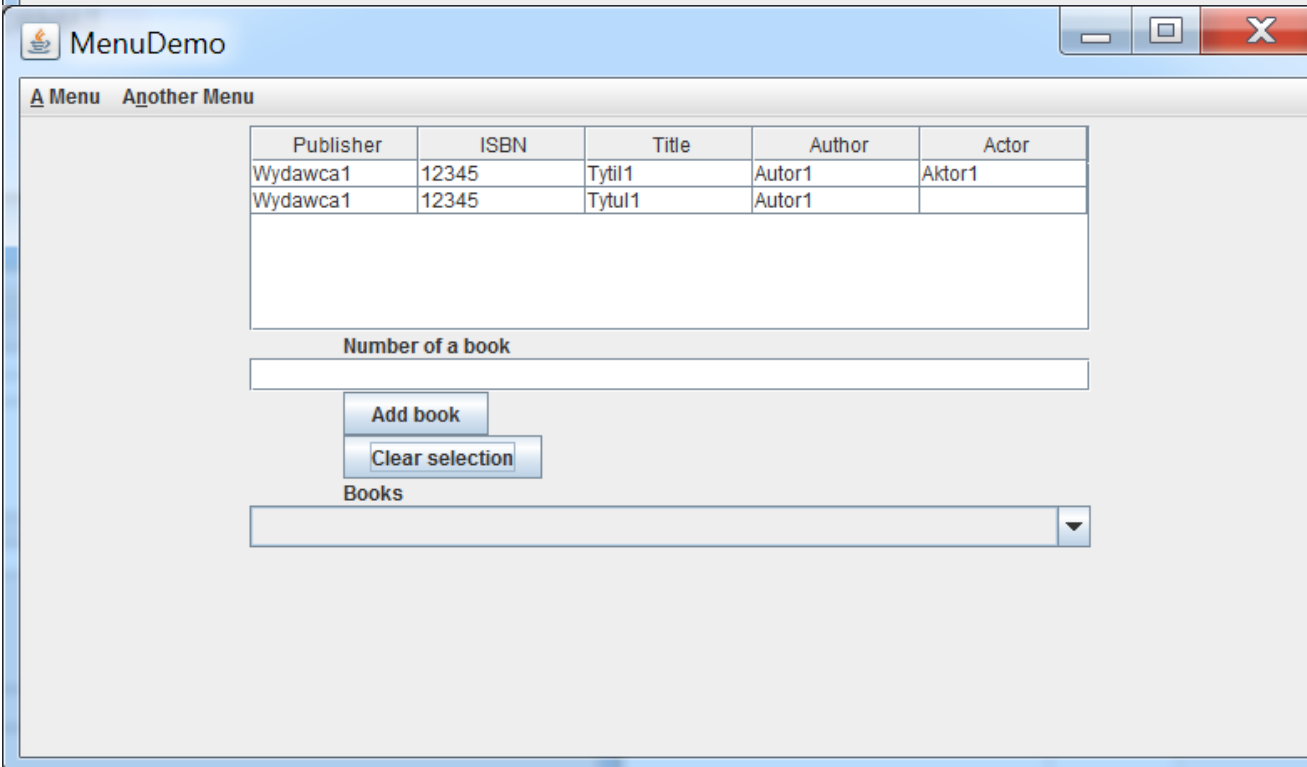
```
Library1_EE (run-deploy) | Java DB Database Process | GlassFish Server
WARN: WELD-001700: Interceptor annotation class javax.ejb.PostActivate n
WARN: WELD-001700: Interceptor annotation class javax.ejb.PrePassivate n
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] public org.gl
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] private org.g
WARN: WELD-000411: Observer method [BackedAnnotatedMethod] org.glassfish
Info: Library1_EE was successfully deployed in 8 633 milliseconds.
```

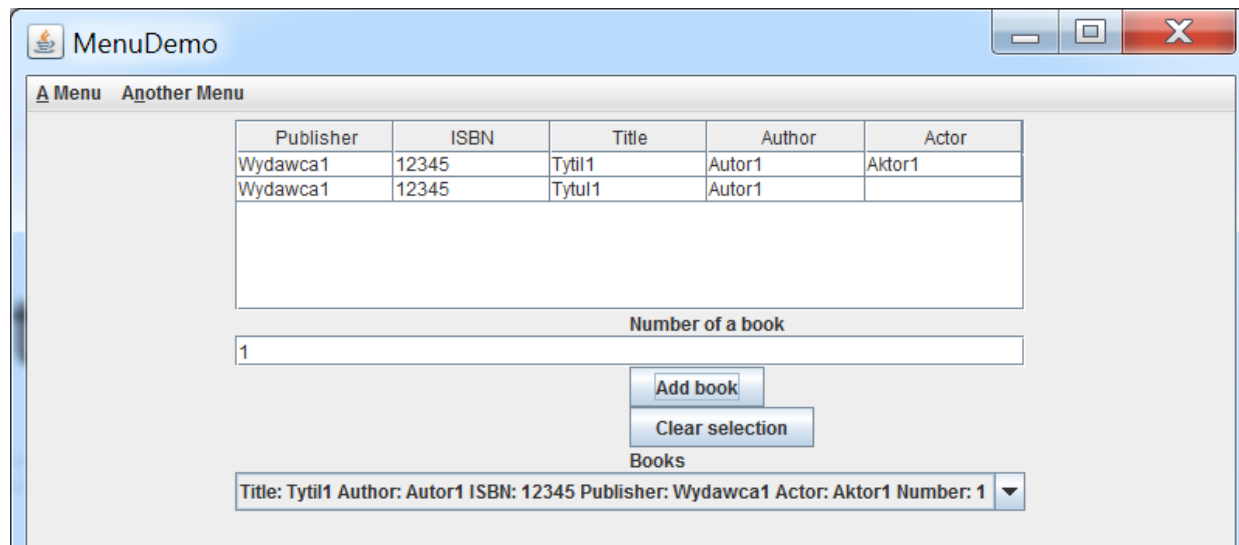
22. Uruchomienie 2 desktopowych aplikacji wielowarstwowych typu EE: 2 razy uruchomić Run na projekcie typu Enterprise Application Client





**23. W dwóch uruchomionych aplikacjach klienckich typu Enterprise istnieje dostęp do wspólnych danych, przechowywanych w komponentcie EJB Session Bean typu Stateless.**





**24. W dwóch uruchomionych aplikacjach klienckich typu Enterprise istnieje dostęp do wspólnych danych, przechowywanych w komponente EJB Session Bean typu Stateless.**

