

Laboratoria 5-7-część 2

Diagramy klas i sekwencji

Przykład

Zofia Kruczkiewicz

Produkt – oprogramowanie na platformie Java EE

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

Warstwa klienta

Klienci aplikacji, aplety, aplikacje i inne elementy z graficznym interfejsem użytkownika

Interakcja z użytkownikiem, urządzenia i prezentacja interfejsu użytkownika

Warstwa prezentacji

Strony JSP, serwlety i inne elementy interfejsu użytkownika

Logowanie, zarządzanie sesją, tworzenie zawartości, formatowania i dostarczanie

Warstwa biznesowa

Komponenty EJB i inne obiekty biznesowe

Logika biznesowa, transakcje, dane i usługi

Warstwa integracji

JMS, JDBC, konektory i połączenia z systemami zewnętrznymi

Adaptory zasobów, systemy zewnętrzne, mechanizmy zasobów, przepływ sterowania

Warstwa zasobów

Bazy danych, systemy zewnętrzne i pozostałe zasoby

Zasoby, dane i usługi zewnętrzne

Identyfikacja klas

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

- Zidentyfikuj zbiór klas, które współpracują ze sobą w celu wykonania poszczególnych czynności
- Określ zbiór zobowiązań każdej klasy
- Rozważ zbiór klas jako całość: **podziel na mniejsze te klasy**, które mają zbyt wiele zobowiązań; **scal w większe te klasy**, które mają zbyt mało zobowiązań
- Rozpatrz sposoby wzajemnej kooperacji tych klas i porozdzielaj ich zobowiązania tak, aby żadna z nich była **ani zbyt złożona ani zbyt prosta**
- **Elementy nieprogramowe (urządzenia)** przedstaw w postaci klasy i odróżnij go za pomocą własnego stereotypu; jeśli ma on oprogramowanie, może być traktowany jako węzeł diagramu klas w celu rozwijania tego oprogramowania
- Zastosuj typy pierwotne (tabele, wyliczenia, typy proste np. boolean itp)

Identyfikacja związków: zależność (Dependency)

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

Modelowanie zależności

- Utworzyć zależności między klasą z operacją, a klasą użytą jako parametr tej operacji
- Stosuj **zależności tylko wtedy**, gdy modelowany związek nie jest strukturalny

Identyfikacja związków: generalizacja czyli dziedziczenie (Generalization)

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

- Ustaliwszy zbiór klas poszukaj **zobowiązań, atrybutów i operacji wspólnych** dla co najmniej dwóch klas
- Przenieś te wspólne zobowiązania, atrybuty i operacje do klasy bardziej ogólnej; jeśli to konieczne, utwórz nową klasę, do której zostaną przypisane te właśnie byty (uwagaż z wprowadzaniem zbyt wielu poziomów generalizacji)
- Zaznacz, że klasy szczegółowe dziedziczą po klasie ogólnej, to znaczy uwzględnij uogólnienia biegnące od każdego potomka do bardziej ogólnego przodka
- Stosuj uogólnienia tylko wtedy, gdy masz do czynienia ze związkiem „jest rodzajem”; **dziedziczenie wielobazowe często można zastąpić agregacją**
- Wystrzegaj się wprowadzania cyklicznych uogólnień
- **Utrzymuj uogólnienia w pewnej równowadze**; krata dziedziczenia nie powinna być zbyt głęboka (pięć lub więcej poziomów już budzi wątpliwości) ani zbyt szeroka (lepiej wprowadzić pośrednie klasy abstrakcyjne)

Identyfikacja związków strukturalnych: powiązanie (Association) , agregacja (Aggregation)

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

- Rozważ, czy w wypadku każdej pary klas jest konieczne przechodzenie od obiektów jednej z nich do obiektów drugiej
- Rozważ, czy w wypadku każdej pary klas jest konieczna inna interakcja między obiektami jednej z nich a obiektami drugiej niż tylko przekazywanie ich jako parametrów; jeśli tak, **uwzględnij powiązanie między tymi klasami**, w przeciwnym wypadku **jest to zależność użycia**. Ta metoda identyfikacji powiązań jest oparta na zachowaniu
- Dla każdego powiązania określ **liczebność** (szczególnie wtedy, kiedy nie jest to 1 - wartość domyślna) i nazwy ról (ponieważ ułatwiają zrozumienie modelu)
- Jeśli jedna z powiązanych klas stanowi strukturalną lub organizacyjną całość w porównaniu z klasami z drugiego końca związku, które wyglądają jak części, zaznacz przy niej specjalnym symbolem, że chodzi o **agregację**.
- **Stosuj powiązania głównie wtedy, kiedy między obiektami zachodzą związki strukturalne**

Identyfikacja wzorców projektowych (wstęp do wykładu 6)

- 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)

Przykład identyfikacji klas

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

Przykład z wykładu 3, Przykład 3 z wykładu 4 (cd)

Diagram przypadków użycia (wykład 4 część 1, przykład 3)

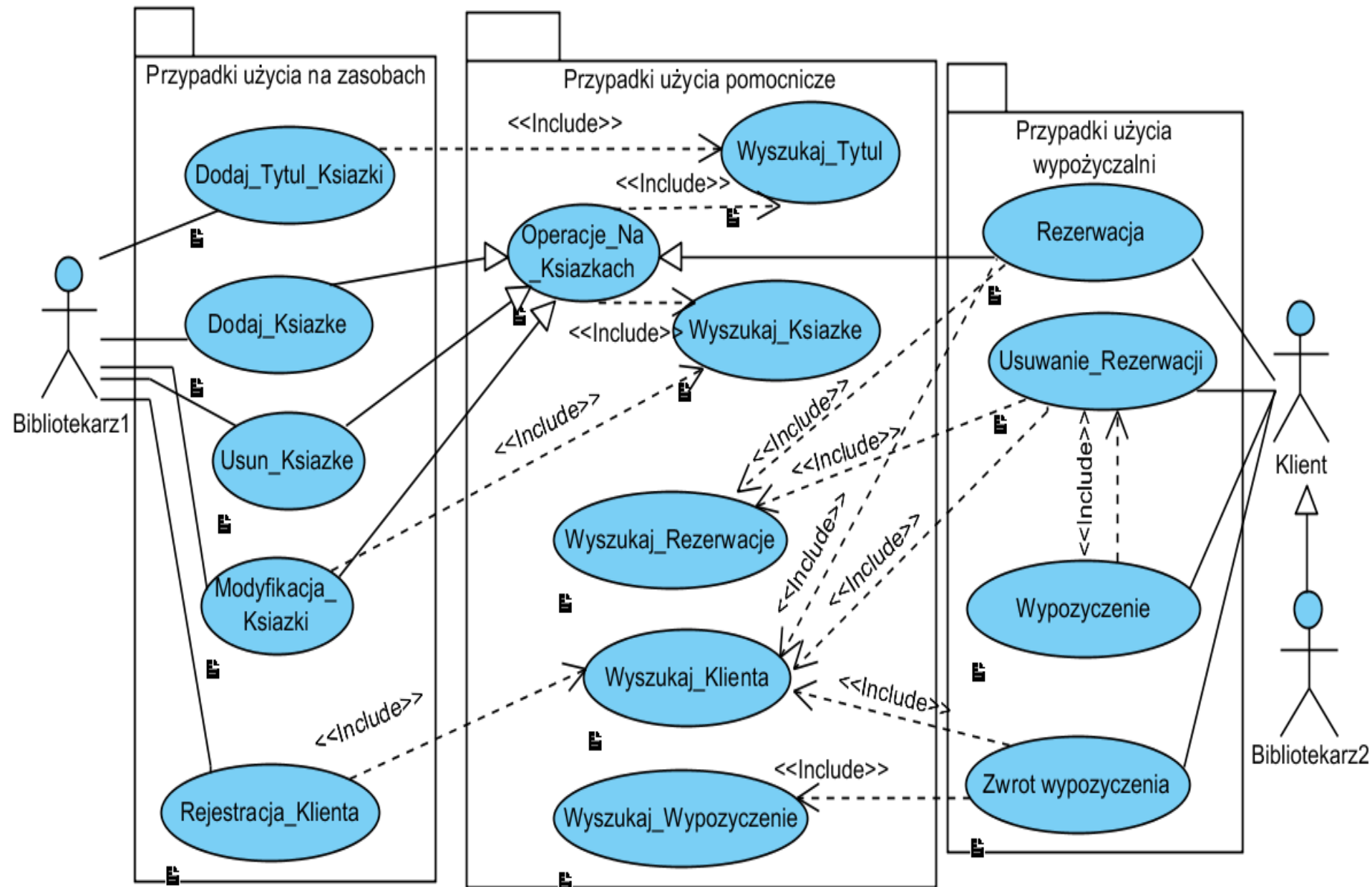
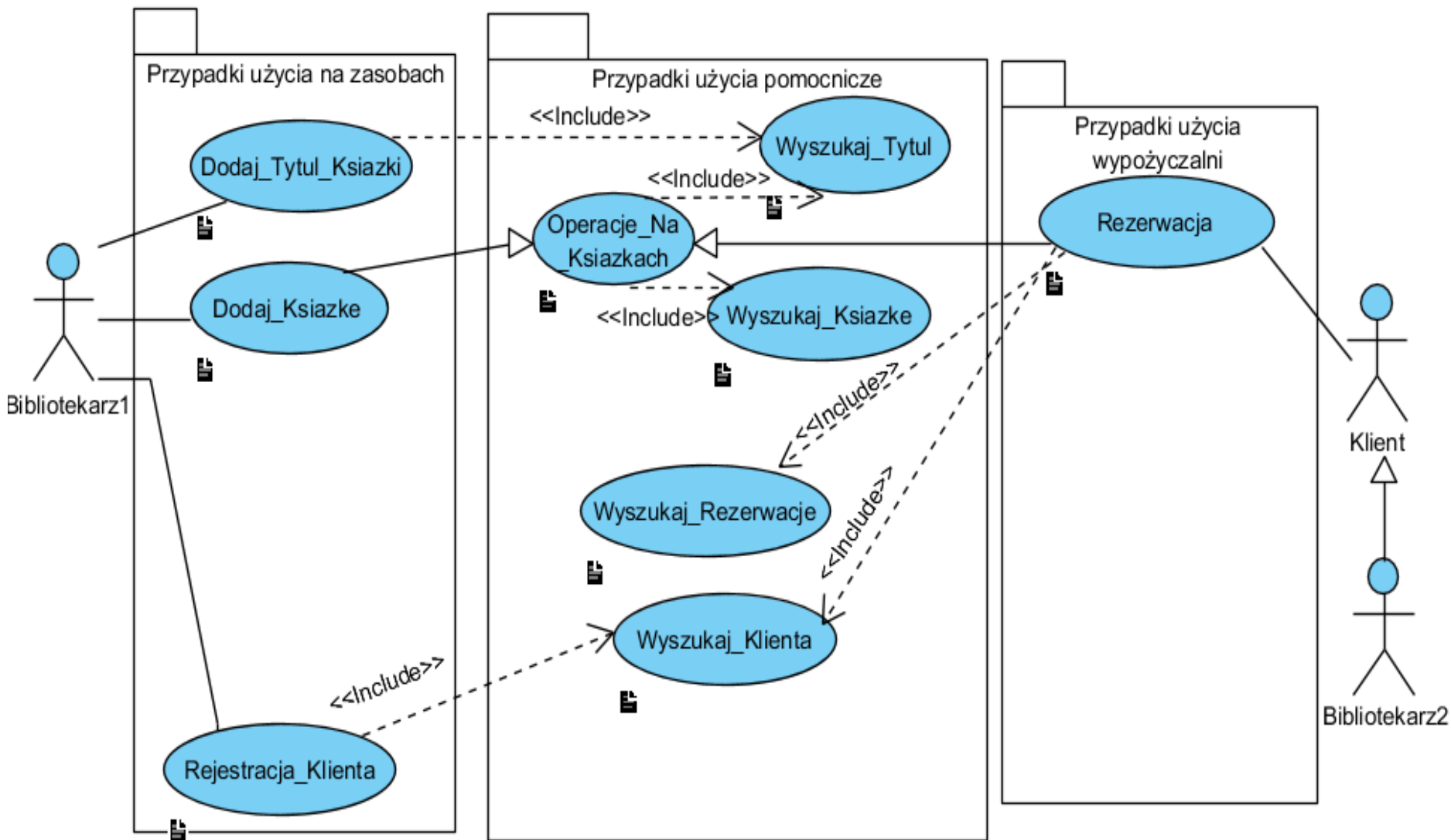


Diagram przypadków użycia (wykład 4 część 1, przykład 3) – wybrany fragment



Identyfikacja klas – etap 1

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

Analiza ta powinna być umieszczona w zakładce **General/Description** formularza udostępnionego za pomocą pozycji **Open Specification...**, wybranej po kliknięciu na powierzchnię tworzonego diagramu klas w ramach projektu.

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

Identyfikacja klas – etap 2

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

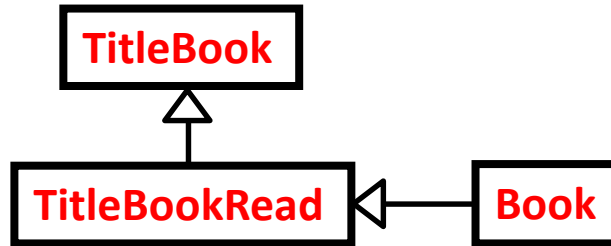
- Wykryto **dziedziczenie** w właściwościach tytułów książek, które określają, które książki są nagrane, a które są drukowane
Zdefiniowano klasę pochodną:
 - **TitleBookRead** typu „**Entity**”, która dziedziczy od klasy **TitleBook** i zawiera informację o aktorze, która czyta książkę na nagraniu (PU: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

Analiza wspólności i zmienności - identyfikacja typów relacji

Oszacowania dla przyjętego modelu powiązań:

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

Oszacowania dla częściowo równoważnego modelu dziedziczenia:



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

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

Analiza zmienności (c.d)

- **Wykryto związki:**

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

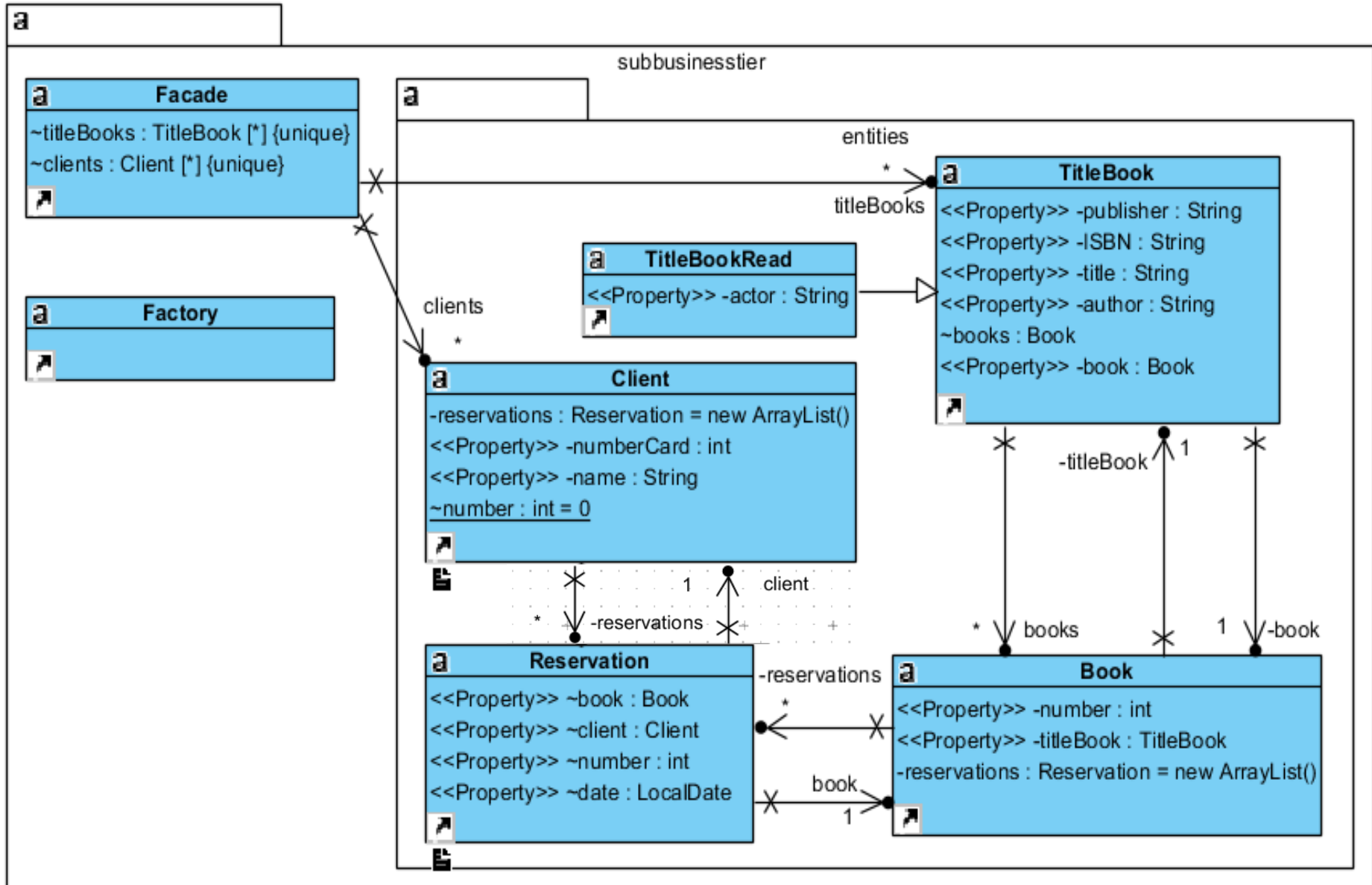
Analiza zmienności (c.d)

- **Zastosowano wzorzec strukturalny typu Fasada:**
 - klasę fasadową **Facade** typu „**Control**” do oddzielenia przetwarzania obiektów typu „**Entity**” od pozostałej części systemu

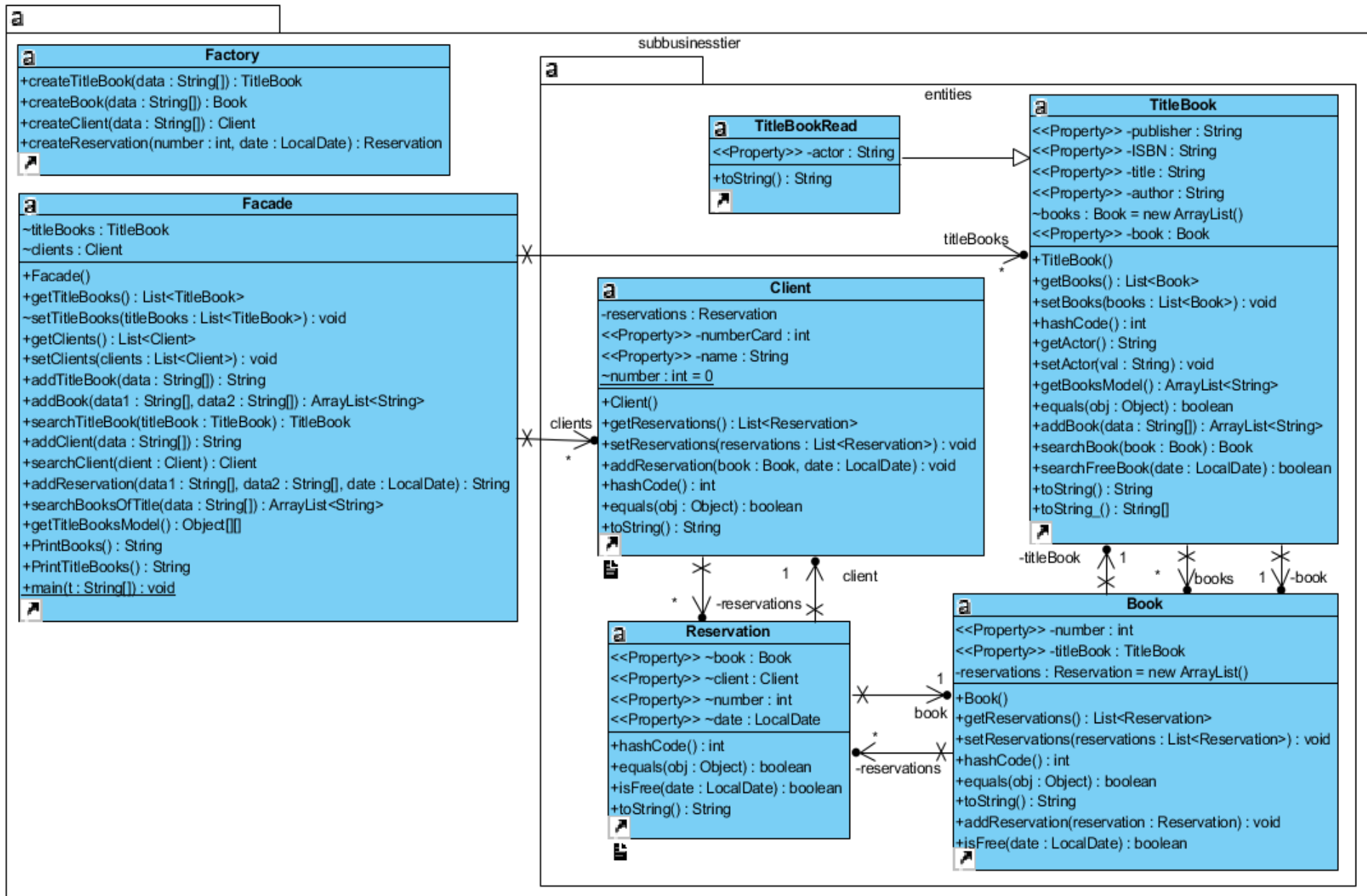
- **Zastosowano wzorzec wytwórczy typu Fabryka**
 - klasę typu „**Control**” jako fabrykę obiektów (**Factory**) do tworzenia różnych typów produktów – czyli obiektów typu **TitleBook**, **TitleBookRead**, **Book**, **Reservation**, **Client**

Wynik

Początkowa definicja diagramu klas – zdefiniowano powiązania między klasami



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



Klasa Facade udostępnia metody logiki biznesowej – implementacja przypadków użycia wywoływanych przez aktorów na diagramie przypadków użycia

```
package subbusinesssier;  
import java.time.LocalDate;  
import java.time.Month;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import subbusinesssier.entities.Client;  
import subbusinesssier.entities.TitleBook;  
public class Facade {  
    List<TitleBook> titleBooks;  
    List<Client> clients;  
    public Facade() { }  
    public List<TitleBook> getTitleBooks() { }  
    public void setTitleBooks(List<TitleBook> titleBooks) { }  
    public List<Client> getClients() { }  
    public void setClients(List<Client> clients) { }
```

public TitleBook searchTitleBook(TitleBook titleBook) {}

PU Operacje_Na_Ksiazkach

public Client searchClient(Client client) {}

public String addClient(String data[]) {}

PU Rejestracja_Klienta

public String addTitleBook(String data[]) {}

PU Dodaj_Tytul_Ksiazki

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

PU Dodaj_Ksiazke

public String addReservation(String data1[], String data2[], LocalDate date) {}

PU Rezerwacja

// pomocnicze metody

public ArrayList<String> searchBooksOfTitle(String data[]) {}

public Object[][] getTitleBooksModel() {}

public String PrintBooks() {}

public String PrintTitleBooks() {}

public static void main(String t[]) {}

}

Iteracja 1

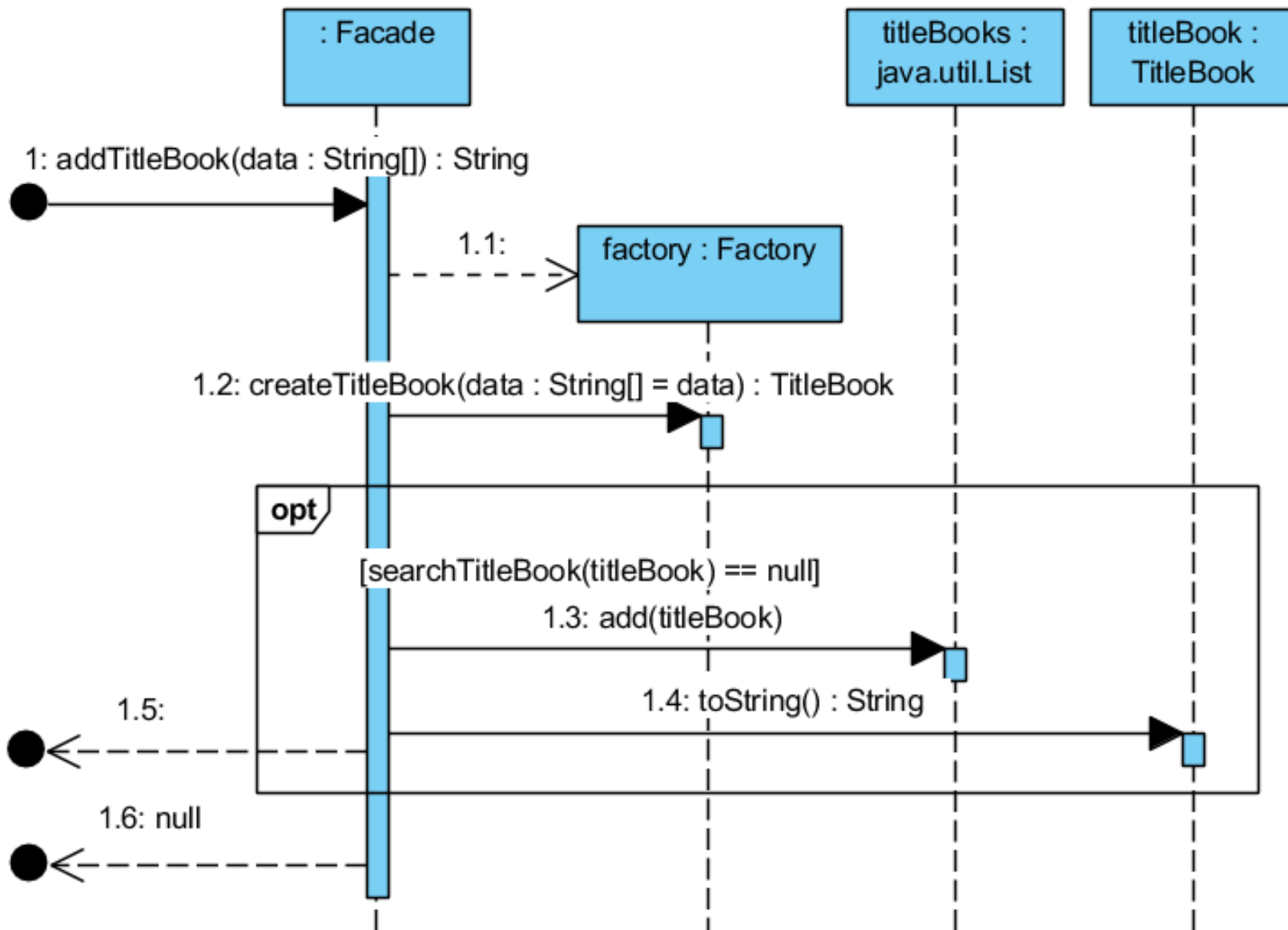
Projekt przypadku użycia

„**Dodaj_Tytul_Ksiazki**”

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

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

`sd subbusinessstier.Facade.addTitleBook(String)`



```
//class Facade
```

```
List<TitleBook> titleBooks;
```

```
List<Client> clients;
```

```
public Facade() {
```

```
    titleBooks = new ArrayList<>();
```

```
    clients = new ArrayList();
```

```
}
```

```
public String addTitleBook(String data[]) {
```

```
    Factory factory = new Factory();
```

```
    TitleBook titleBook = factory.createTitleBook(data);
```

```
    if (searchTitleBook(titleBook) == null) {
```

```
        titleBooks.add(titleBook);
```

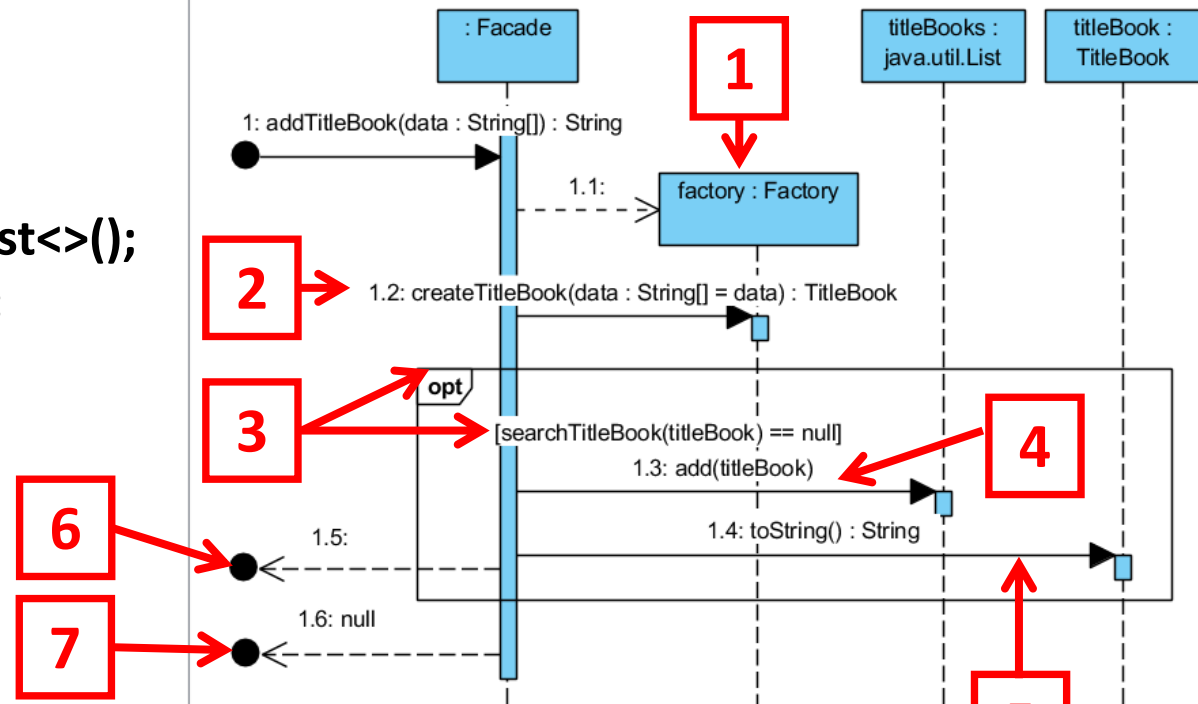
```
        return titleBook.toString();
```

```
    }
```

```
    return null;
```

```
}
```

```
sd subbusinessstier.Facade.addTitleBook(String)
```



```
//1
```

```
//2
```

```
//3
```

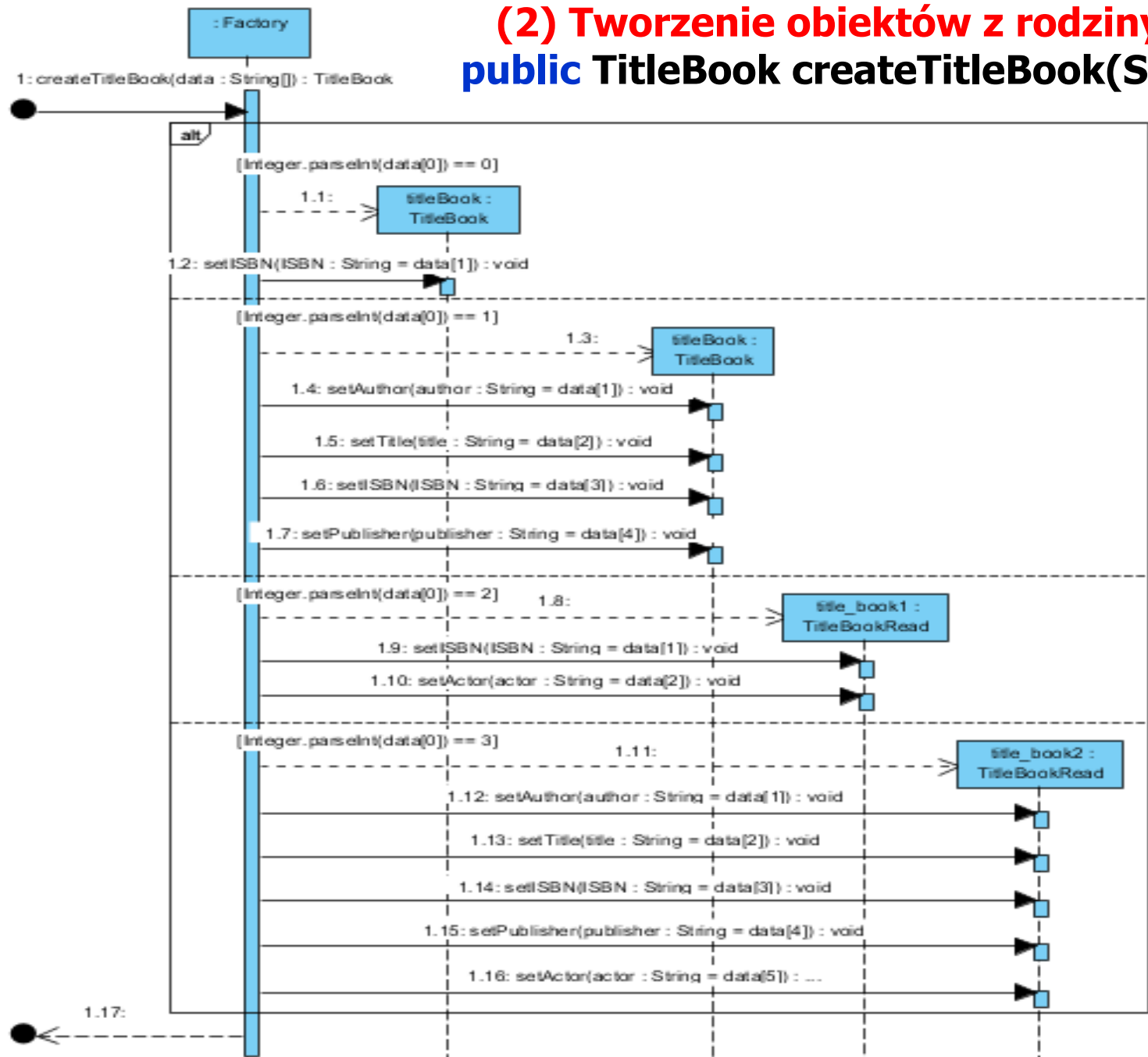
```
//4
```

```
//5, 6
```

```
//7
```

(2) Tworzenie obiektów z rodziny TitleBook

public TitleBook createTitleBook(String data[])



```
public class Factory { //Factory -decyzje na poziomie tworzenia kodu
```

```
public TitleBook createTitleBook(String data[]) {
```

```
TitleBook titleBook = null;
```

```
switch (Integer.parseInt(data[0])) //what_title_book_type
```

```
{
```

```
    case 0:
```

```
        titleBook = new TitleBook(); //TitleBook object for searching
```

```
        titleBook.setISBN(data[1]);
```

```
        break;
```

```
    case 1:
```

```
        titleBook = new TitleBook(); //TTitleBook object for persisting
```

```
        titleBook.setAuthor(data[1]);
```

```
        titleBook.setTitle(data[2]);
```

```
        titleBook.setISBN(data[3]);
```

```
        titleBook.setPublisher(data[4]);
```

```
        break;
```


case 2:

```
TitleBookRead title_book1 = new TitleBookRead();  
                                //TitleBookRead object for searching  
title_book1.setISBN(data[1]);  
title_book1.setActor(data[2]);  
titleBook = title_book1;  
break;
```

case 3:

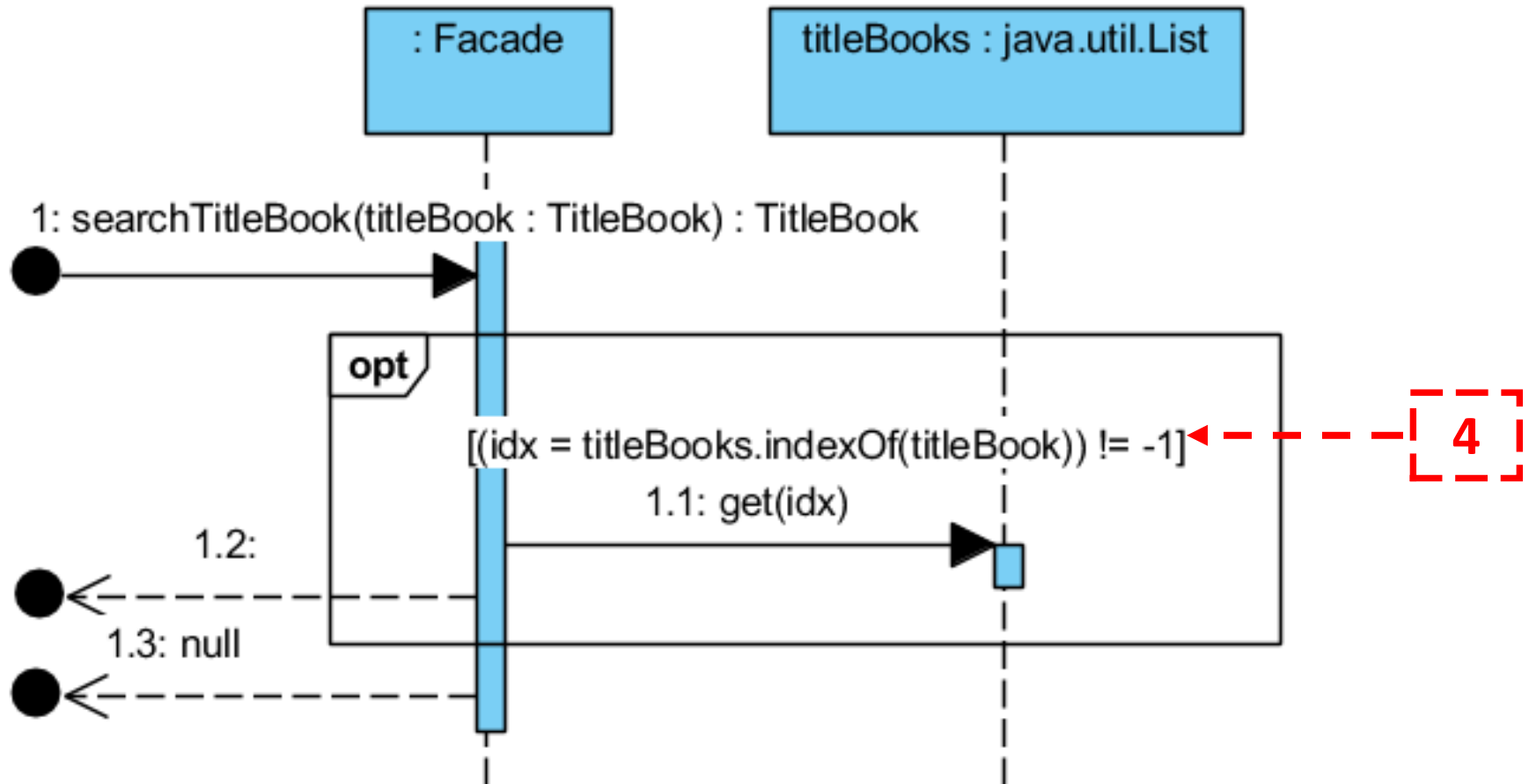
```
TitleBookRead title_book2 = new TitleBookRead();  
                                //TitleBookRead object for persisting  
title_book2.setAuthor(data[1]);  
title_book2.setTitle(data[2]);  
title_book2.setISBN(data[3]);  
title_book2.setPublisher(data[4]);  
title_book2.setActor(data[5]);  
titleBook = title_book2;  
break; }
```

```
return titleBook;
```

```
}
```

(3) Wyszukiwanie obiektu z rodziny TitleBook

sd subbusiness-tier.Facade.searchTitleBook(TitleBook)



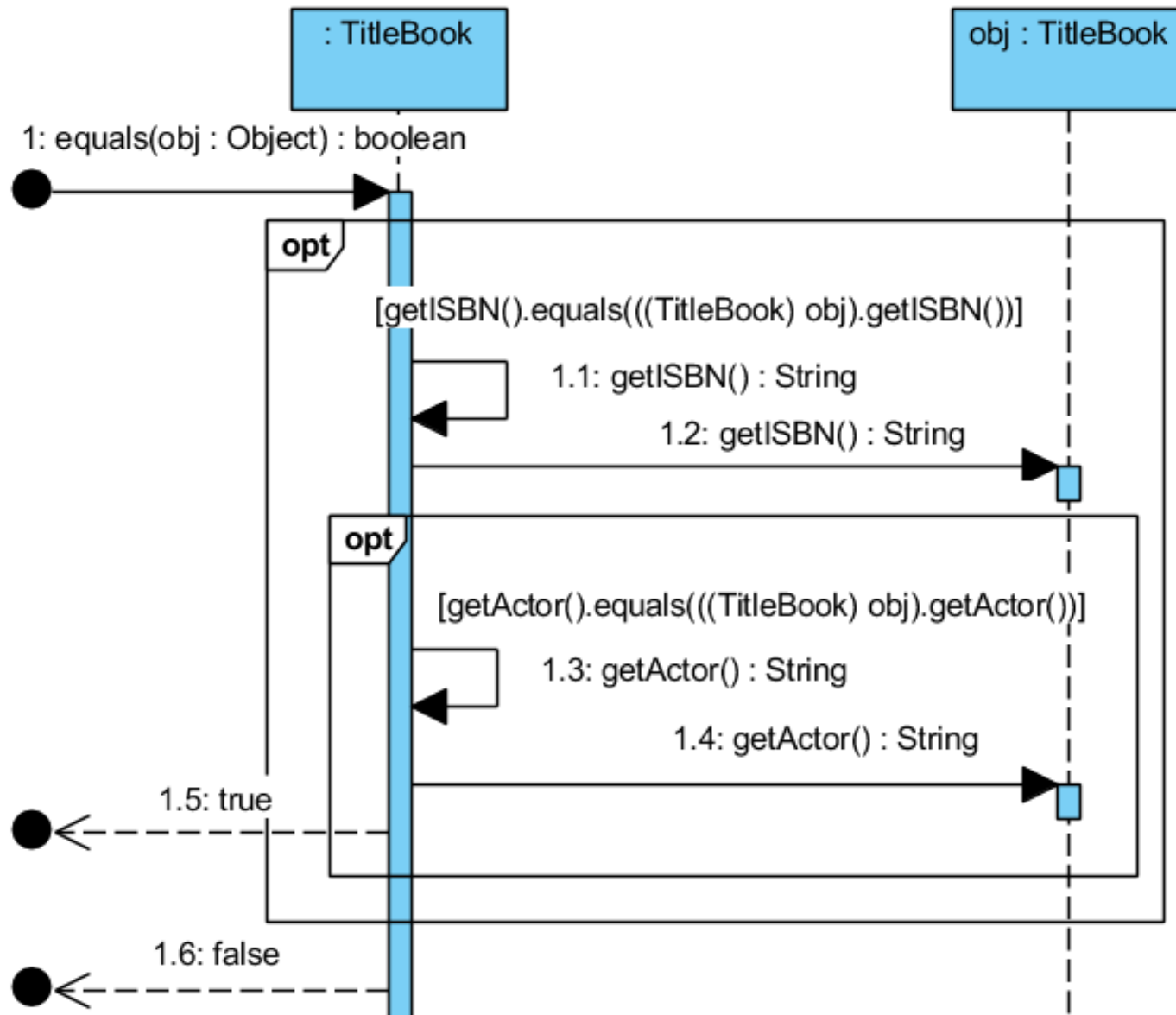
//class Facade

```
public TitleBook searchTitleBook(TitleBook titleBook) {  
    int idx;  
    if ((idx = titleBooks.indexOf(titleBook)) != -1) {  
        return titleBooks.get(idx);  
    }  
    return null;  
}
```

```
public int indexOf(Object o) {  
    if (o == null) {  
        for (int i = 0; i < size; i++)  
            if (elementData[i]==null)  
                return i;  
    } else {  
        for (int i = 0; i < size; i++)  
            if (o.equals(elementData[i]))  
                return i; }  
    return -1; }
```

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

sd subbusinessier.entities.TitleBook.equals(Object)

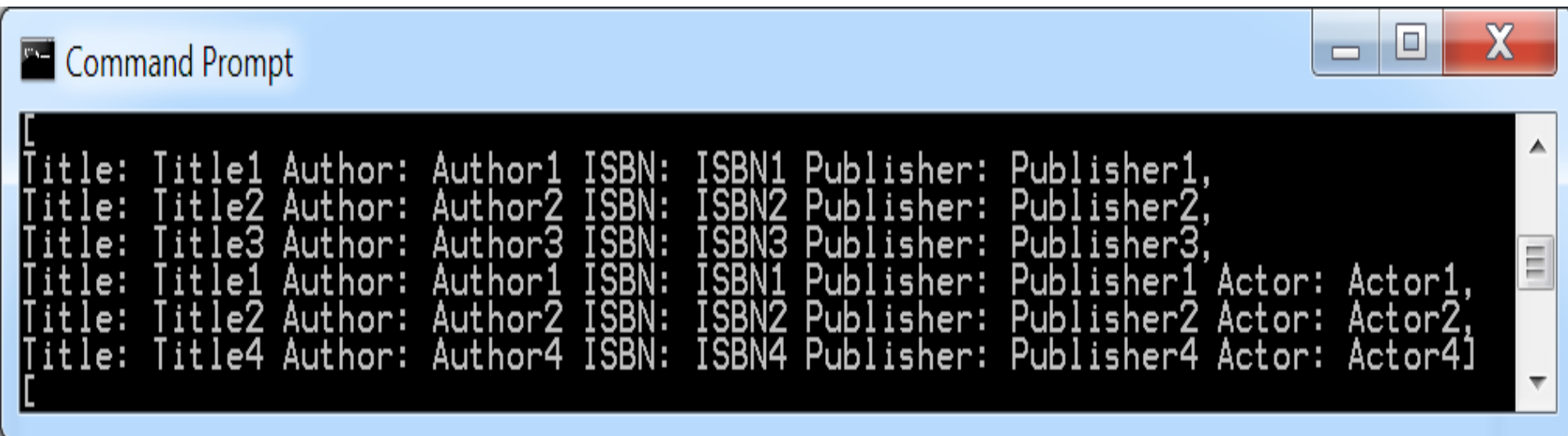


```
//class TitleBook
```

```
@Override
```

```
public boolean equals(Object obj) {  
    boolean result = false;  
    if (getISBN().equals(((TitleBook) obj).getISBN())) {  
        if (getActor().equals(((TitleBook) obj).getActor())) {  
            result = true;  
        }  
    }  
    return result;  
}
```

```
public static void main(String args[])
{
    Facade ap = new Facade();
    String title1[] = {"1", "Author1", "Title1", "ISBN1", "Publisher1"},
        dtitle1[] = {"0", "ISBN1"};
    String title2[] = {"1", "Author2", "Title2", "ISBN2", "Publisher2"},
        dtitle2[] = {"0", "ISBN2"};
    String title3[] = {"1", "Author3", "Title3", "ISBN3", "Publisher3"},
        dtitle3[] = {"0", "ISBN5"};
    String title4[] = {"3", "Author1", "Title1", "ISBN1", "Publisher1", "Actor1"},
        dtitle4[] = {"2", "ISBN1", "Actor1"};
    String title5[] = {"3", "Author2", "Title2", "ISBN2", "Publisher2", "Actor2"};
    String title6[] = {"3", "Author4", "Title4", "ISBN4", "Publisher4", "Actor4"},
        dtitle5[] = {"2", "ISBN4", "Actor4"};
    ap.addTitleBook(title1);      //dodawanie tytułów książek
    ap.addTitleBook(title2);
    ap.addTitleBook(title2);
    ap.addTitleBook(title3);
    ap.addTitleBook(title4);
    ap.addTitleBook(title5);
    ap.addTitleBook(title5);
    ap.addTitleBook(title6);
    String lan = ap.getTitleBooks().toString();
}
```



```
[
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]
[
```

**Dodatek – kolejne iteracje rozwoju tworzenia
oprogramowania
(kontynuacja przykładu)**

Iteracja 2

Projekt przypadku użycia

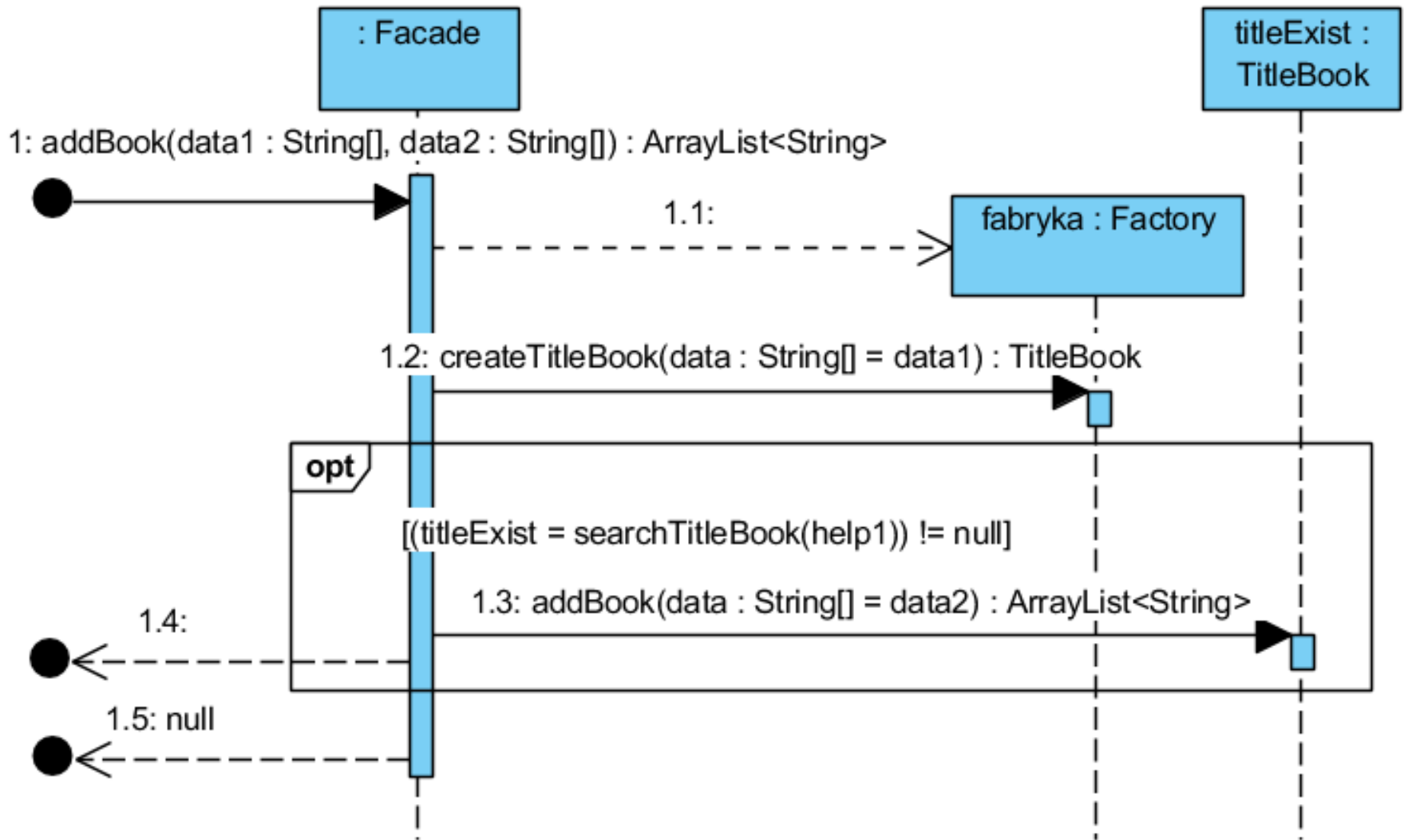
„Dodaj_Ksiazke”

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

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

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

sd subbusinesssier.Facade.addBook(String, String)



```
//class Facade
```

```
List<TitleBook> titleBooks;
```

```
List<Client> clients;
```

```
public Facade() {
```

```
    titleBooks = new ArrayList<>();
```

```
    clients = new ArrayList();
```

```
}
```

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

```
    TitleBook help1, titleExist;
```

```
    Factory fabryka = new Factory();
```

```
    help1 = fabryka.createTitleBook(data1); //metoda
```

```
    //ponownie użyta
```

```
    if ((titleExist = searchTitleBook(help1)) != null) { //metoda
```

```
        return titleExist.addBook(data2); //ponownie użyta
```

```
    }
```

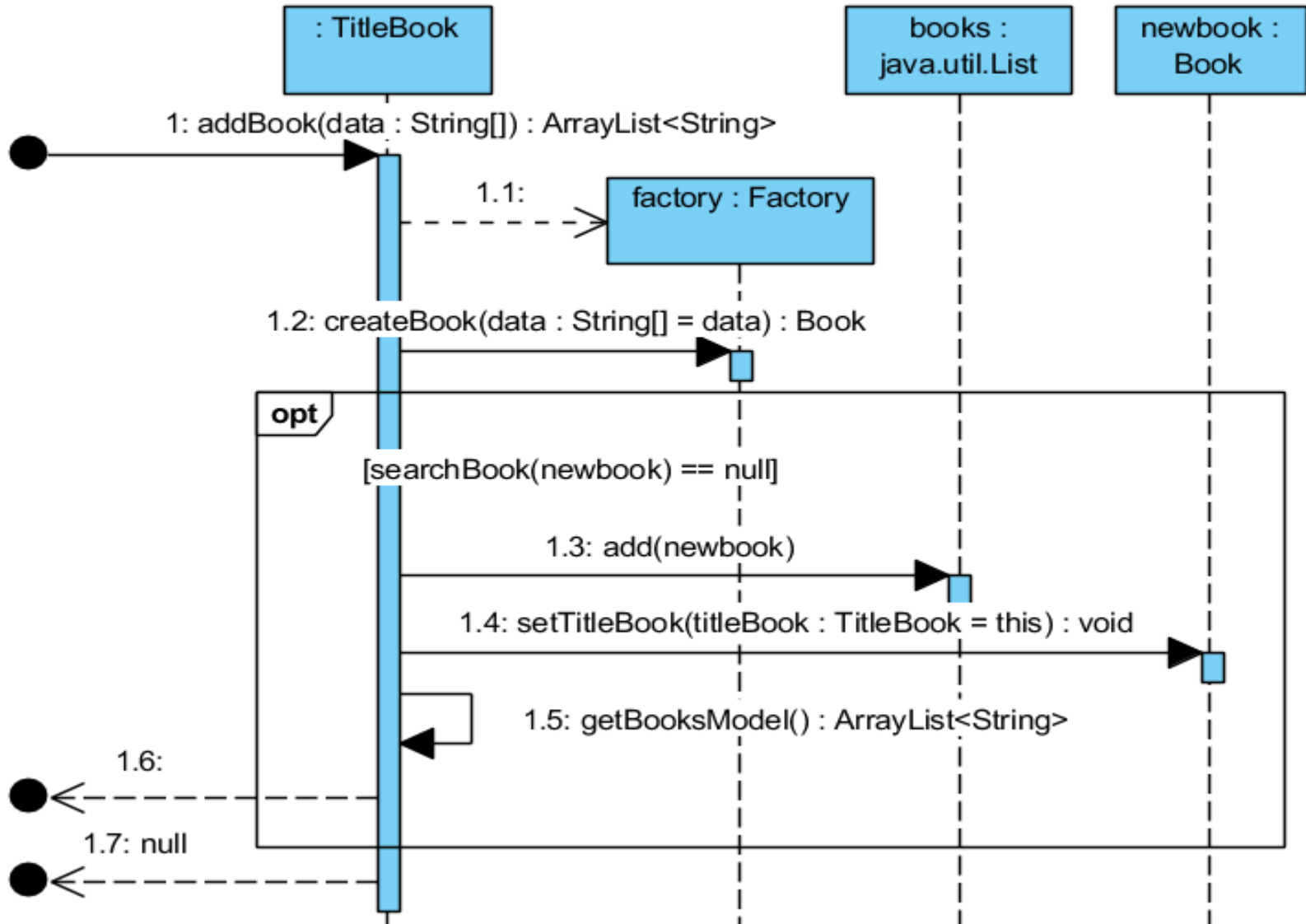
```
    return null;
```

```
}
```

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

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

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



```
//class TitleBook
```

```
List<Book> books;
```

```
public TitleBook() {
```

```
    books = new ArrayList();
```

```
}
```

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

```
    Factory factory = new Factory();
```

```
    Book newbook;
```

```
    newbook = factory.createBook(data);
```

```
    if (searchBook(newbook) == null) {
```

```
        books.add(newbook);
```

```
        newbook.setTitleBook(this);
```

```
        return getBooksModel();
```

```
    }
```

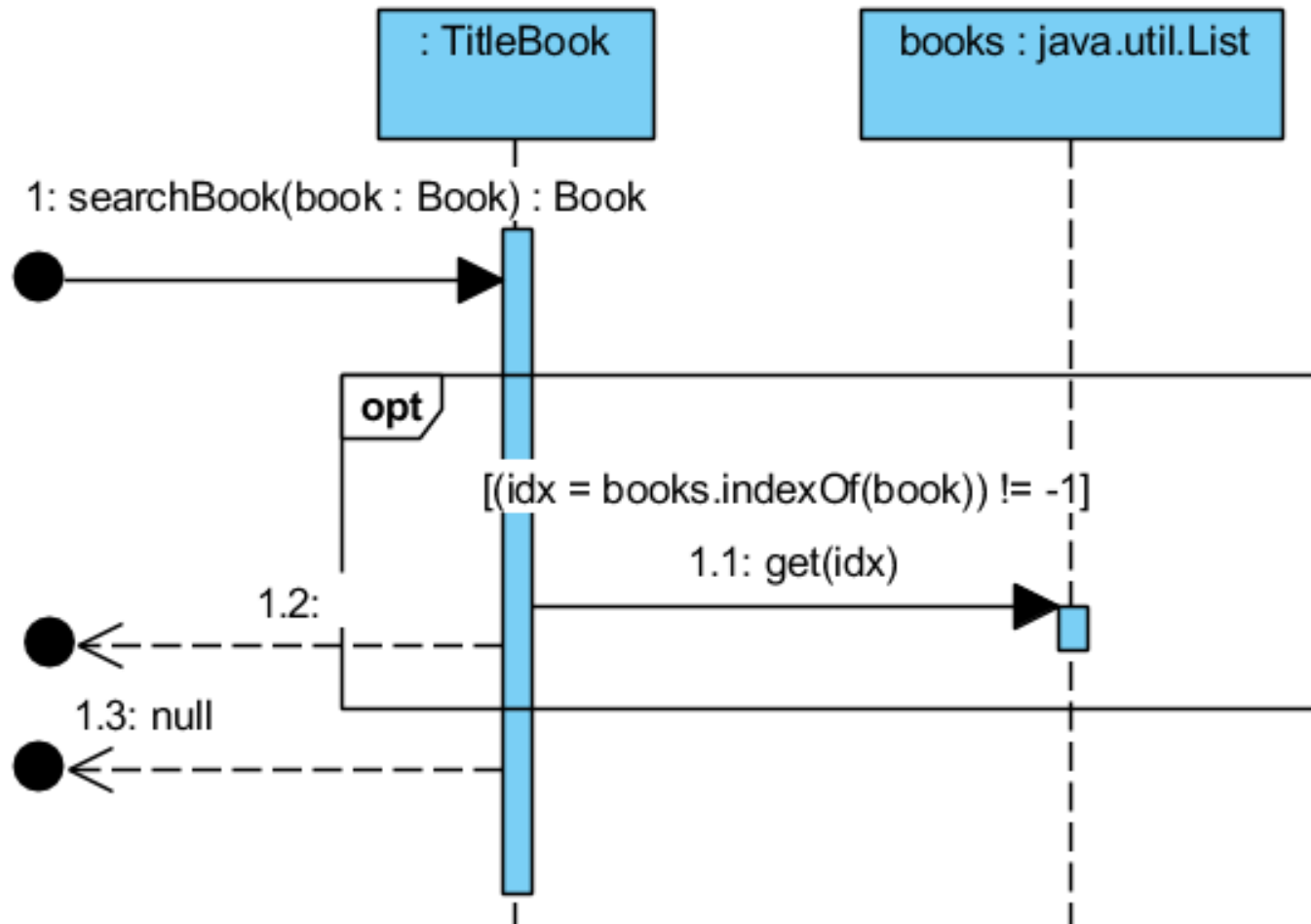
```
    return null;
```

```
}
```

(7) Szukanie książki

public Book searchBook(Book book)

sd subbusinessstier.entities.TitleBook.searchBook(Book)



```
//class TitleBook
```

```
List<Book> books;
```

```
public TitleBook() {
```

```
    books = new ArrayList();
```

```
}
```

```
public Book searchBook(Book book) {
```

```
    int idx;
```

```
    if ((idx = books.indexOf(book)) != -1) {
```

```
        return (Book) books.get(idx);
```

```
}
```

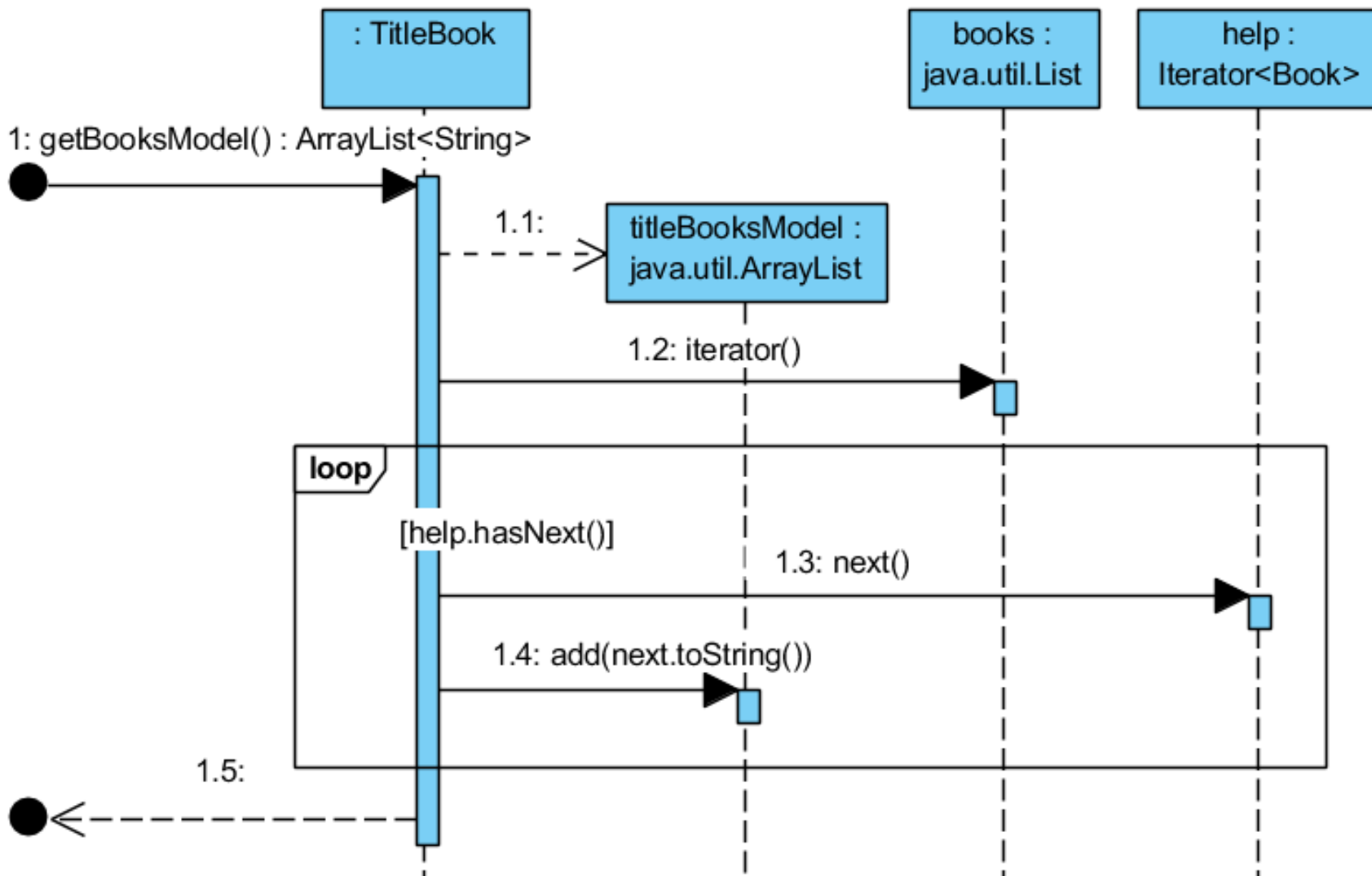
```
    return null;
```

```
}
```

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

`public ArrayList<String> getBooksModel()`

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




```
//class TitleBook
```

```
List<Book> books;
```

```
public TitleBook() {  
    books = new ArrayList();  
}
```

```
public ArrayList<String> getBooksModel() {  
    ArrayList<String> titleBooksModel = new ArrayList<>();  
    Iterator<Book> help = books.iterator();  
    while (help.hasNext()) {  
        Book next = help.next();  
        titleBooksModel.add(next.toString());  
    }  
    return titleBooksModel;  
}
```

```
public static void main(String args[])
{
    Facade ap = new Facade();
    String title1[] = {"1", "Author1", "Title1", "ISBN1", "Publisher1"},
        dtitle1[] = {"0", "ISBN1"};
    String title2[] = {"1", "Author2", "Title2", "ISBN2", "Publisher2"},
        dtitle2[] = {"0", "ISBN2"};
    String title3[] = {"1", "Author3", "Title3", "ISBN3", "Publisher3"},
        dtitle3[] = {"0", "ISBN5"};
    String title4[] = {"3", "Author1", "Title1", "ISBN1", "Publisher1", "Actor1"},
        dtitle4[] = {"2", "ISBN1", "Actor1"};
    String title5[] = {"3", "Author2", "Title2", "ISBN2", "Publisher2", "Actor2"};
    String title6[] = {"3", "Author4", "Title4", "ISBN4", "Publisher4", "Actor4"},
        dtitle5[] = {"2", "ISBN4", "Actor4"};
    ap.addTitleBook(title1);           //dodawanie tytułów książek
    ap.addTitleBook(title2);
    ap.addTitleBook(title2);
    ap.addTitleBook(title3);
    ap.addTitleBook(title4);
    ap.addTitleBook(title5);
    ap.addTitleBook(title5);
    ap.addTitleBook(title6);
    String lan = ap.getTitleBooks().toString();
}
```

// cd metody main – dodawanie książek

```
System.out.println(lan);  
String book1[] = {"0", "1"};  
String book2[] = {"0", "2"};  
ArrayList<String> pom = ap.addBook(dttitle1, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book2);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle3, book2);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle4, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle5, book2);  
if (pom != null) System.out.print(pom);
```

```
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

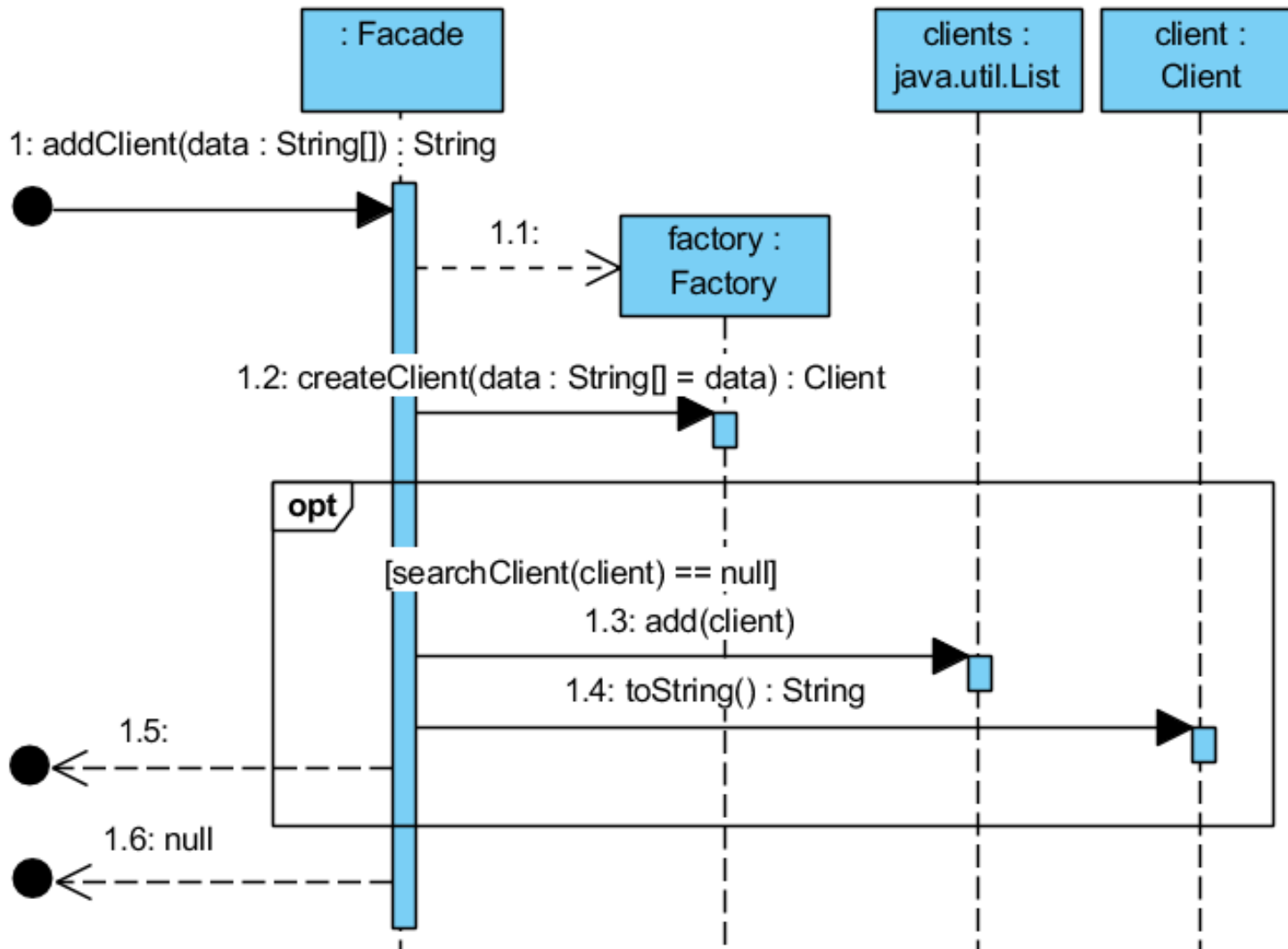
„ **Rejestracja_Klienta** ”

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

(9) Wstawianie nowego klienta wypożyczalni

public String addClient(String data[])

sd subbusinesssier.Facade.addClient(String)



//class Facade

List<TitleBook> titleBooks;

List<Client> clients;

public Facade() {

titleBooks = new ArrayList<>();

clients = new ArrayList();

}

public String addClient(String data[]) {

Factory factory = new Factory();

Client client = factory.createClient(data);

if (searchClient(client) == null) {

clients.add(client);

return client.toString();

}

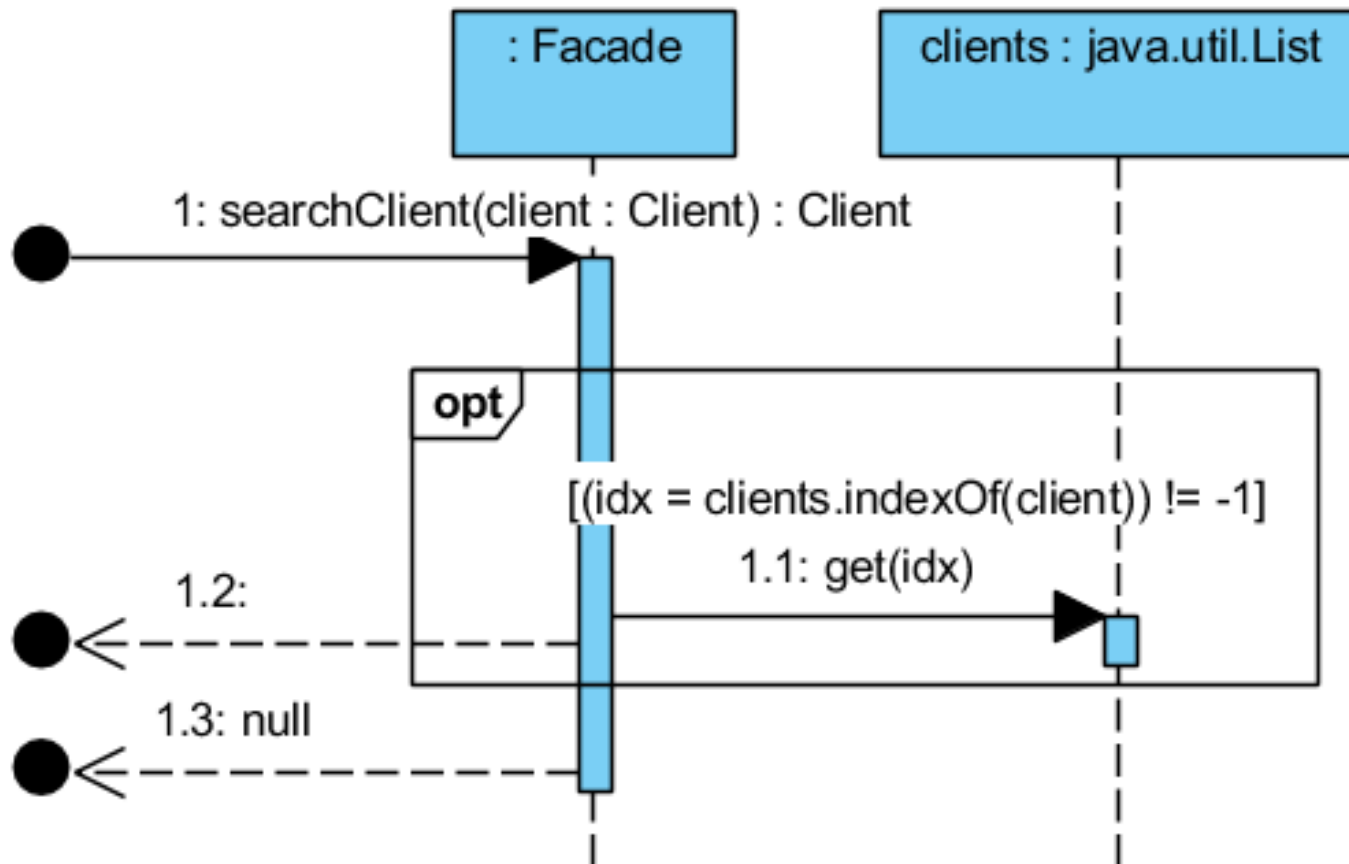
return null;

}

(10) Szukanie klienta

public Client searchClient(Client client)

sd subbusinessstier.Facade.searchClient(Client)




```
//class Facade
```

```
List<TitleBook> titleBooks;
```

```
List<Client> clients;
```

```
public Facade() {
```

```
    titleBooks = new ArrayList<>();
```

```
    clients = new ArrayList();
```

```
}
```

```
public Client searchClient(Client client) {
```

```
    int idx;
```

```
    if ((idx = clients.indexOf(client)) != -1) {
```

```
        return clients.get(idx);
```

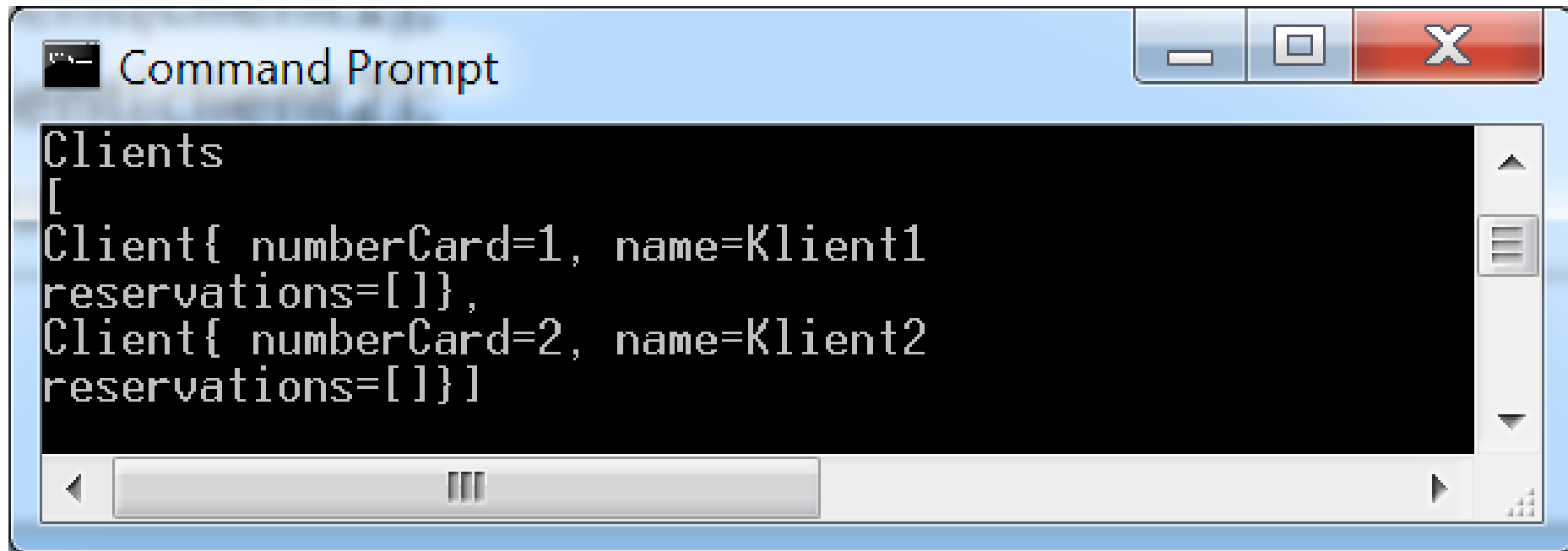
```
    }
```

```
    return null;
```

```
}
```

//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=[]}]
```

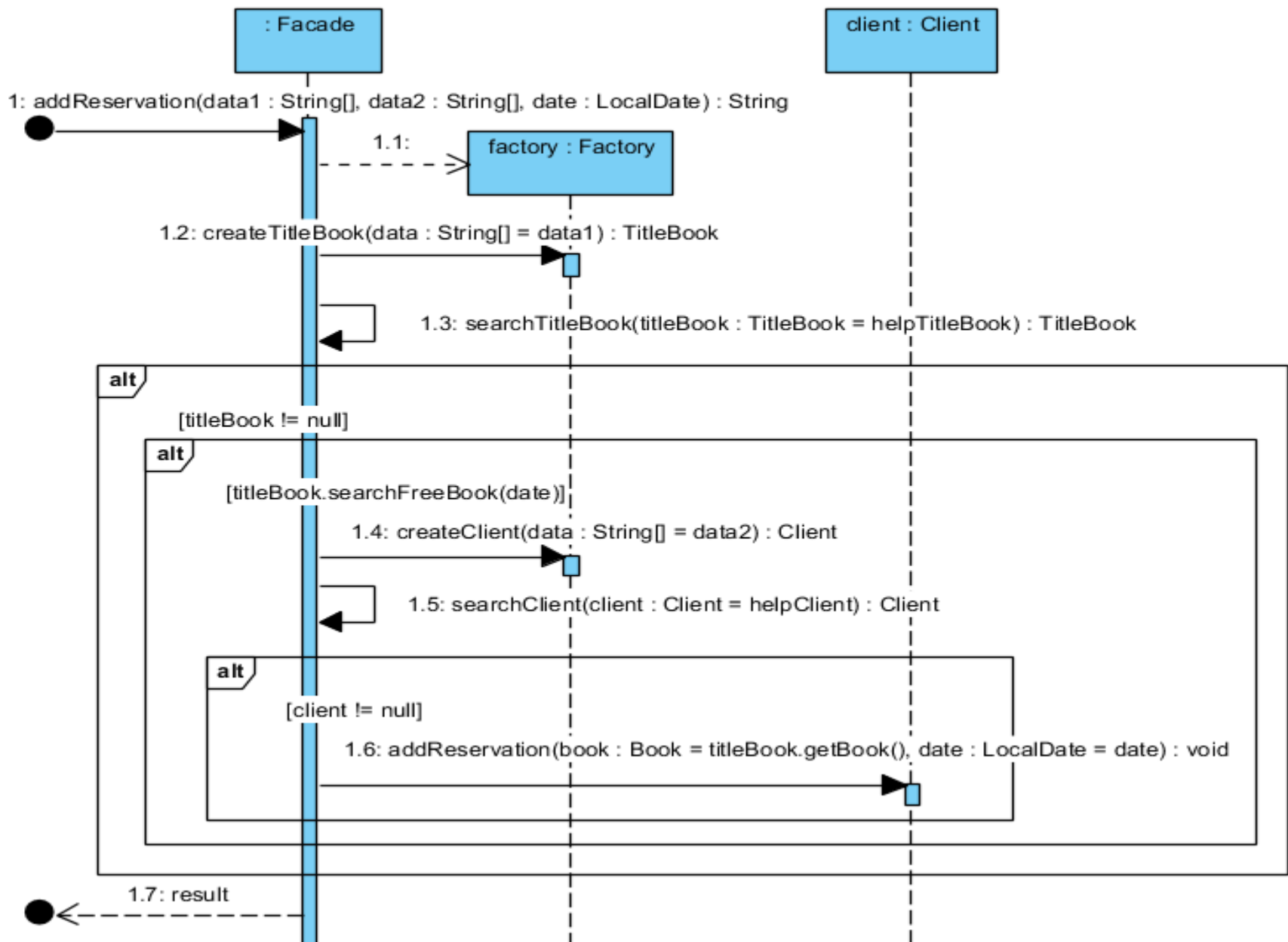
Iteracja 4

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.

(11) Rezerwacja książki: `public String addReservation(String data1[], String data2[], LocalDate date)`

`sd subbusinesssier.Facade.addReservation(String, String, LocalDate)`



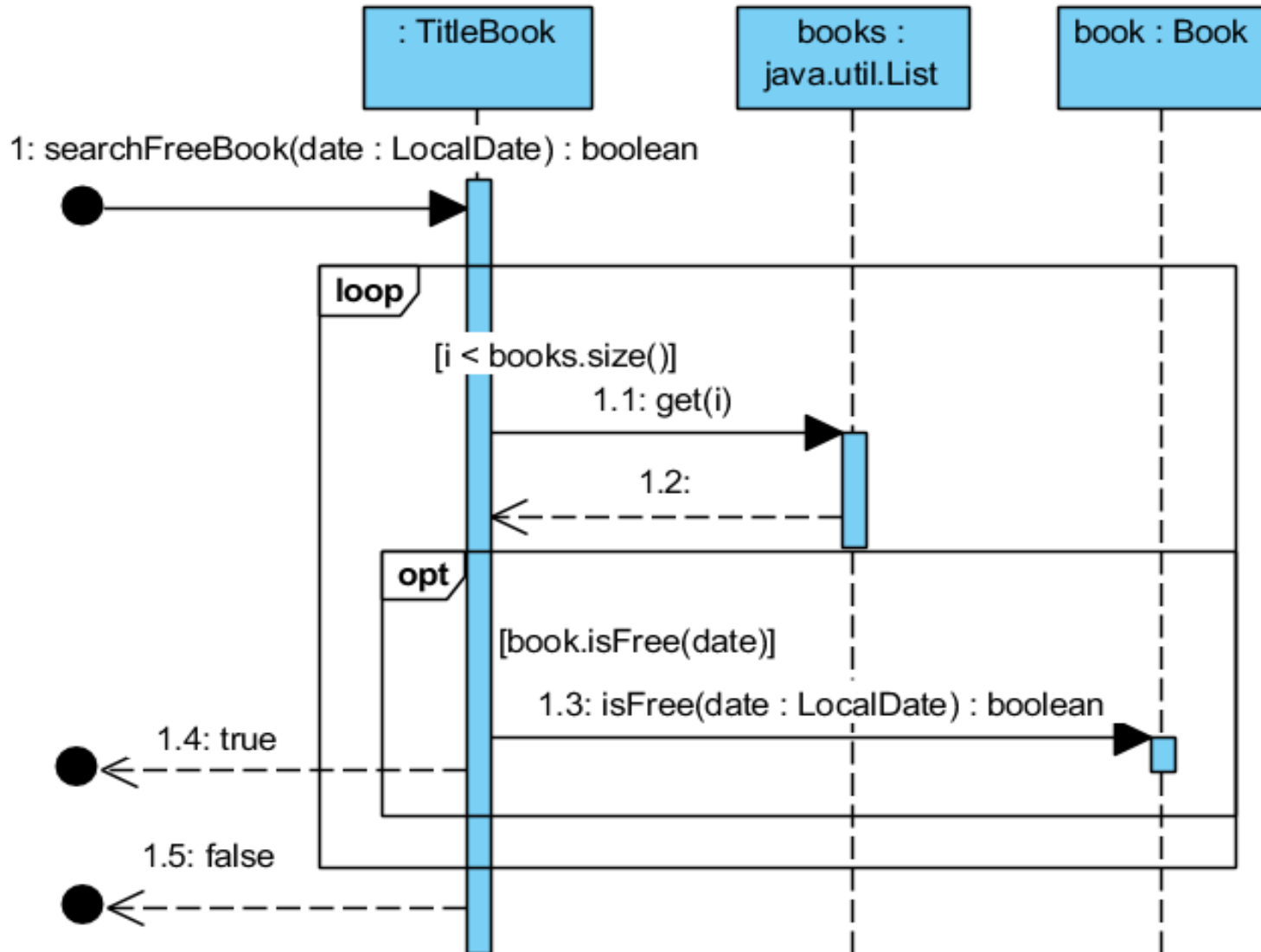
//class Facade

```
public String addReservation(String data1[], String data2[], LocalDate date) {  
    String result;  
    Factory factory = new Factory();  
    TitleBook helpTitleBook = factory.createTitleBook(data1), titleBook;  
                                                                    // metoda użyta ponownie  
    titleBook = this.searchTitleBook(helpTitleBook); // metoda użyta ponownie  
    if (titleBook != null)  
        if (titleBook.searchFreeBook(date)) { //nowa metoda  
            Client helpClient = factory.createClient(data2), client; // metoda użyta  
                                                                    //ponownie  
            client = this.searchClient(helpClient); // metoda użyta ponownie  
            if (client != null) {  
                client.addReservation(titleBook.getBook(), date); //nowa metoda  
                result = "reserved"; //wykonana rezerwacja  
            } else result = "no such a client"; //brak klienta  
        } else result = "no free book"; //brak wolnej książki  
    else result = "no such a title"; //brak tytułu  
    return result; //zwrócenie informacji o wyniku rezerwacji  
}
```

(12) Wyszukiwanie wolnej książki do rezerwacji

public boolean searchFreeBook(LocalDate date)

sd subbusinessier.entities.TitleBook.searchFreeBook(LocalDate) /



```
//class TitleBook
```

```
List<Book> books;
```

```
public TitleBook() {
```

```
    books = new ArrayList();
```

```
}
```

```
private Book book; //atrybut book przechowuje obiekt typu  
                    //Book wyszukany do rezerwacji
```

```
public boolean searchFreeBook(LocalDate date) {
```

```
    for (int i = 0; i < books.size(); i++) {
```

```
        book = books.get(i);
```

```
        if (book.isFree(date))
```

```
            return true;
```

```
    }
```

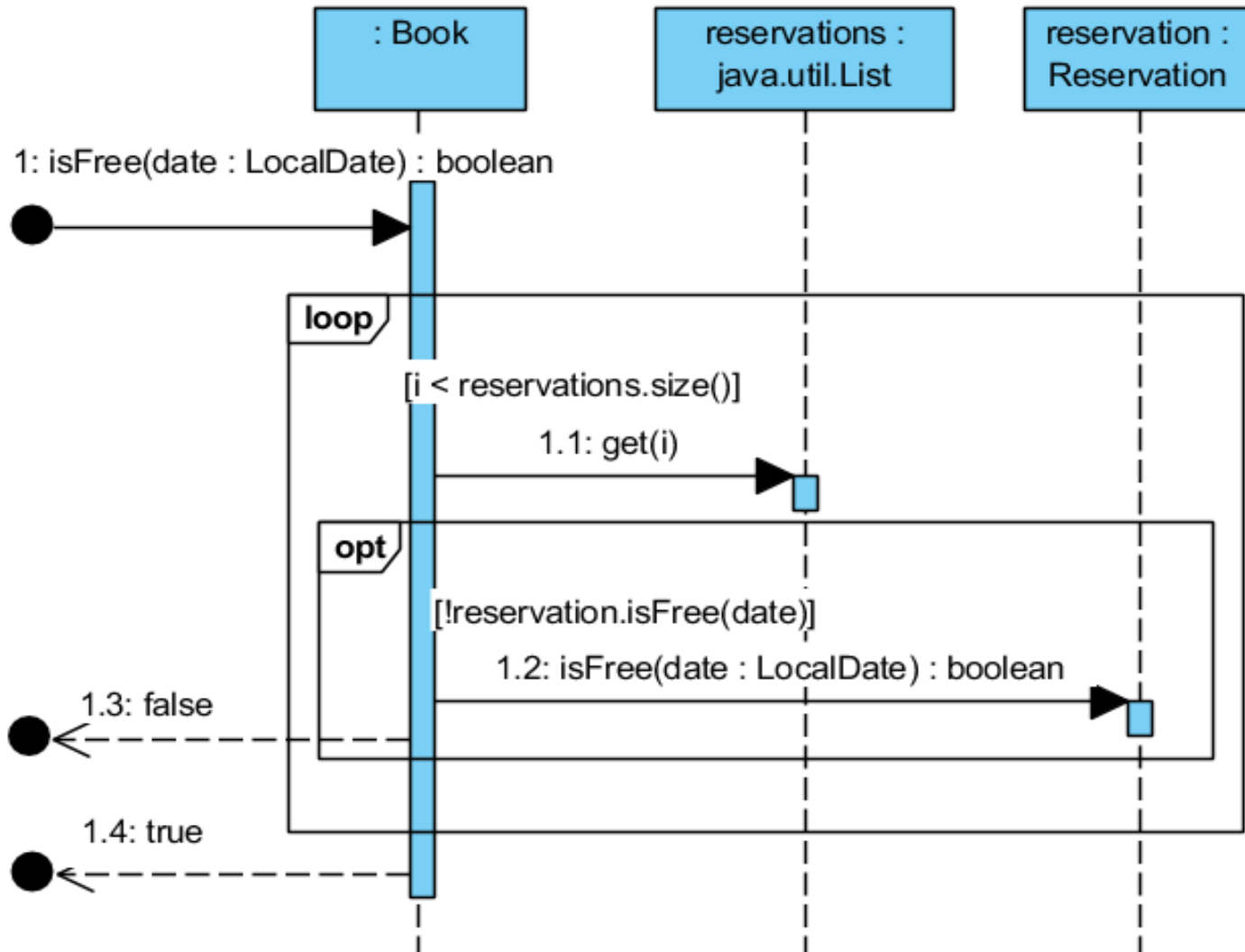
```
    return false;
```

```
}
```

(13) Sprawdzenie przez książkę, czy ma wolny termin rezerwacji

public boolean isFree(LocalDate date)

sd subbusinessstier.entities.Book.isFree(LocalDate) /




```
//class Book
```

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

```
public Book() {
```

```
    reservations = new ArrayList();
```

```
}
```

```
public boolean isFree(LocalDate date) {
```

```
    Reservation reservation;
```

```
    for (int i = 0; i < reservations.size(); i++) {
```

```
        reservation = reservations.get(i);
```

```
        if (!reservation.isFree(date)) {
```

```
            return false;
```

```
        }
```

```
    }
```

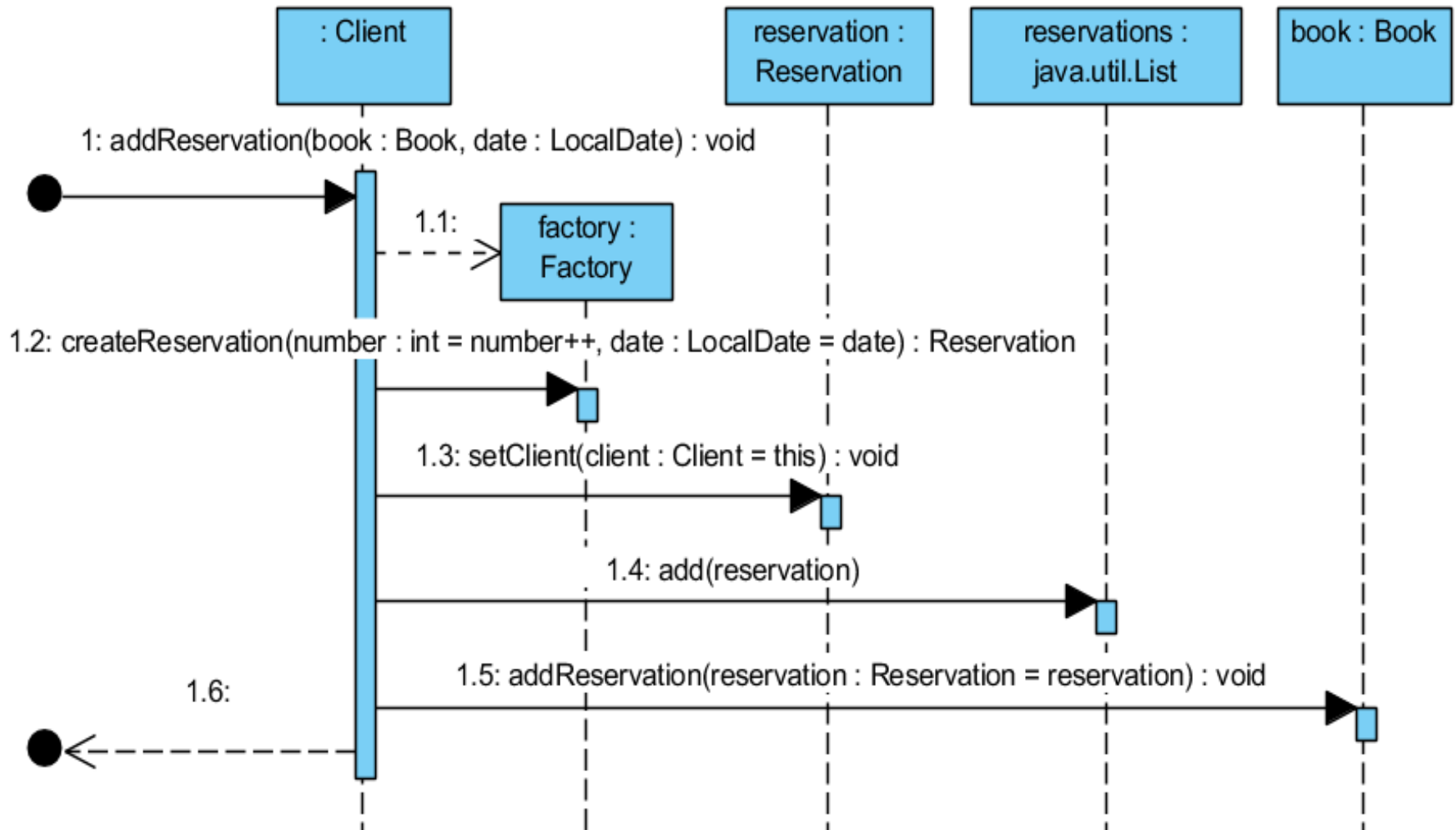
```
    return true;
```

```
}
```

(14) Wykonanie rezerwacji przez obiekt typu Client – 1-y etap

public void addReservation(Book book, LocalDate date)

sd subbusinessstier.entities.Client.addReservation(Book, LocalDate)

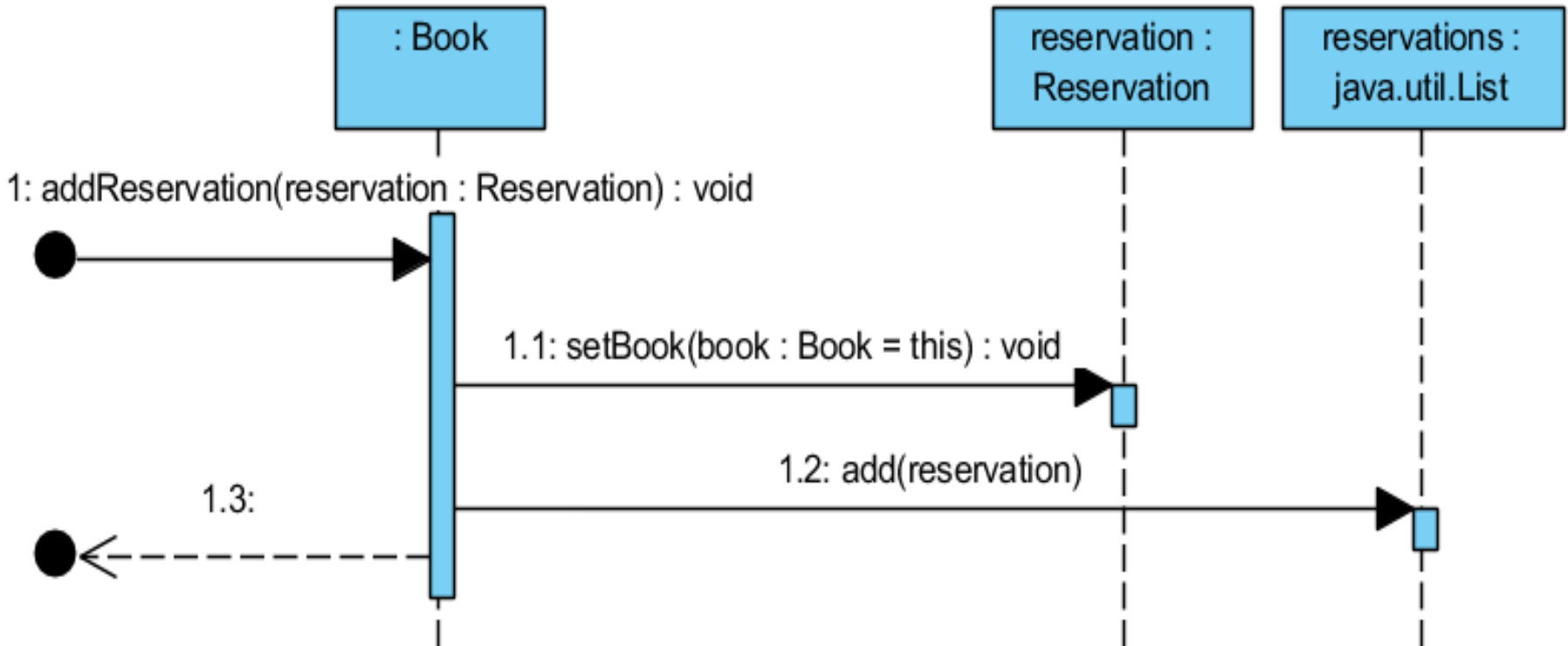


```
                //class Client
private List<Reservation> reservations;
public Client() {
    reservations=new ArrayList();
}

public void addReservation(Book book, LocalDate date)
{
    Factory factory=new Factory();
    Reservation reservation=
        factory.createReservation(number++, date);
    reservation.setClient(this);
    reservations.add(reservation);
    book.addReservation(reservation);
}
```

(15) Wykonanie rezerwacji przez obiekt typu Book – 2-i etap `public void addReservation(Reservation reservation)`

`sd subbusinessstier.entities.Book.addReservation(Reservation)`



```
// class Book
```

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

```
public Book() {
```

```
    reservations = new ArrayList();
```

```
}
```

```
public void addReservation(Reservation reservation) {
```

```
    reservation.setBook(this);
```

```
    reservations.add(reservation);
```

```
}
```

```
public static void main(String args[])
{
    Facade ap = new Facade();
    String title1[] = {"1", "Author1", "Title1", "ISBN1", "Publisher1"},
        dtitle1[] = {"0", "ISBN1"};
    String title2[] = {"1", "Author2", "Title2", "ISBN2", "Publisher2"},
        dtitle2[] = {"0", "ISBN2"};
    String title3[] = {"1", "Author3", "Title3", "ISBN3", "Publisher3"},
        dtitle3[] = {"0", "ISBN5"};
    String title4[] = {"3", "Author1", "Title1", "ISBN1", "Publisher1", "Actor1"},
        dtitle4[] = {"2", "ISBN1", "Actor1"};
    String title5[] = {"3", "Author2", "Title2", "ISBN2", "Publisher2", "Actor2"};
    String title6[] = {"3", "Author4", "Title4", "ISBN4", "Publisher4", "Actor4"},
        dtitle5[] = {"2", "ISBN4", "Actor4"};
    ap.addTitleBook(title1); //dodawanie tytułów książek
    ap.addTitleBook(title2);
    ap.addTitleBook(title2);
    ap.addTitleBook(title3);
    ap.addTitleBook(title4);
    ap.addTitleBook(title5);
    ap.addTitleBook(title5);
    ap.addTitleBook(title6);
    String lan = ap.getTitleBooks().toString();
}
```

//dodawanie książek

```
System.out.println(lan);  
String book1[] = {"0", "1"};  
String book2[] = {"0", "2"};  
ArrayList<String> pom = ap.addBook(dttitle1, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle2, book2);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle3, book2);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle4, book1);  
if (pom != null) System.out.print(pom);  
pom = ap.addBook(dttitle5, book2);  
if (pom != null) System.out.print(pom);
```

```
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]
```


//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);
```

//dodawanie rezerwacji

```
System.out.println("\nReservations");  
LocalDate[] dates = {LocalDate.of(2018, Month.JANUARY, 20),  
                        LocalDate.of(2018, Month.MARCH, 20)};  
System.out.println(ap.addReservation(dttitle1, dclient1, dates[0]));  
System.out.println(ap.addReservation(dttitle1, dclient2, dates[0]));  
System.out.println(ap.addReservation(dttitle2, dclient2, dates[0]));  
System.out.println(ap.addReservation(dttitle2, dclient3, dates[1]));  
System.out.println(ap.addReservation(dttitle2, dclient1, dates[0]));  
System.out.println(ap.addReservation(dttitle2, dclient3, dates[0]));  
System.out.println("\nClients");  
System.out.println(ap.clients);
```

//rezerwacje klientów

}

}

```
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}]}]
```

Rezultat – diagram klas uzyskany w procesie projektowania

