Laboratorium 8
Diagramy aktywności

Zofia Kruczkiewicz

Modelowanie zachowania obiektów za pomocą diagramów aktywności.

Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności - porównanie

Modelowanie zachowania obiektów za pomocą diagramów aktywności.

Cel laboratoriów:

Modelowanie procesów biznesowych procesów realizowanych przez tworzone oprogramowanie w celu zautomatyzowania procesów „świata rzeczywistego” – kontynuacja tworzenia modelu przypadków użycia z wykorzystaniem diagramów czynności (aktywności) (wykład4, instrukcja bieżąca).

Uwaga: Należy rozwijać projekt UML, wykonany podczas lab3-7.

Definiowane zachowania wybranych przypadków użycia za pomocą diagramów czynności – kontynuacja tworzenia modelu przypadków użycia (wg bieżącej instrukcji). Wybrane przypadki użycia muszą realizować procesy biznesowe „ świata rzeczywistego” z p.1. Grupa dwuosobowa realizuje diagram dla jednego złożonego przypadku użycia. Złożony przypadek użycia powinien zawierać relacje <<include>>, lub/i <<extend>>, lub/i <<use**>>. Scenariusze poszczególnych torów powinny odpowiadać scenariuszom modelowanych za pomocą diagramów sekwencji (z lab5-7);**

Uwagi:

1. Należy zastosować elementy diagramów czynności przedstawione na wykładzie 4
2. Diagramy powinny zawierać tzw. partycje (tory).
3. W diagramie czynności reprezentującym scenariusze przypadku użycia głównego i powiązanych podanymi wyżej relacjami należy partycje **zdefiniować jako typy obiektów (klasy) używanych w tworzonym oprogramowaniu i realizujących logikę biznesową scenariuszy**. Partycja wywołująca logikę biznesową umieszczoną w kolejnych partycjach powinna reprezentować interfejs graficzny użytkownika np. GUI, Warstwa prezentacji itp.
4. Przejścia łączące elementy typu: Object Node, Central Buffer Node oraz Data Store Node z innymi elementami diagramu powinny być typu Object Flow. Przejścia pomiędzy pozostałymi elementami diagramu powinny być typu Control Flow. **Pozostale informacje podano w Dodatku do iinstrukcji (str.27-28)**

Modelowanie zachowania obiektów za pomocą diagramów aktywności

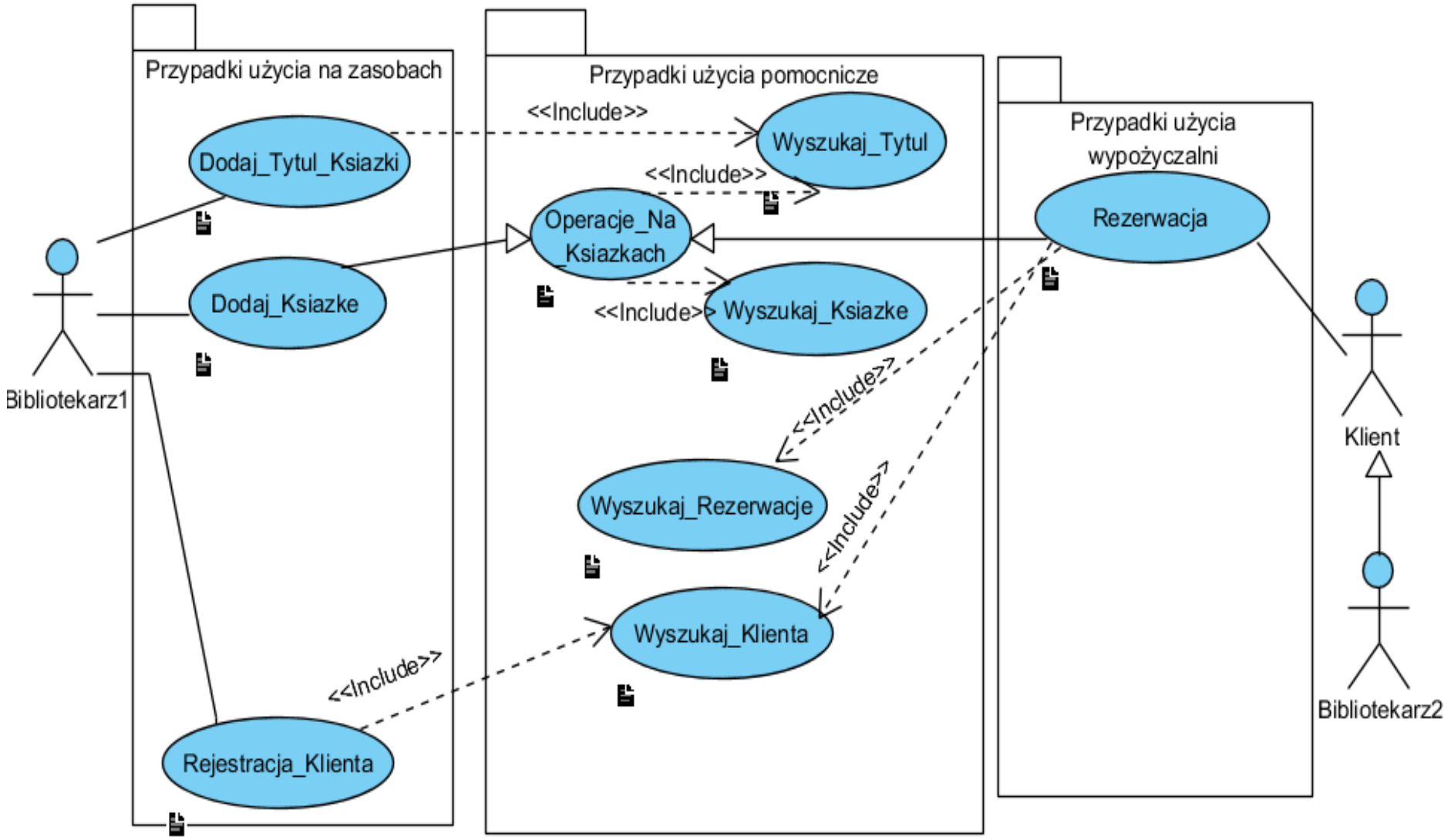
Przykłady z wykładów 4-6

Przykład tworzenia diagramu aktywności w środowisku Visual Paradigm

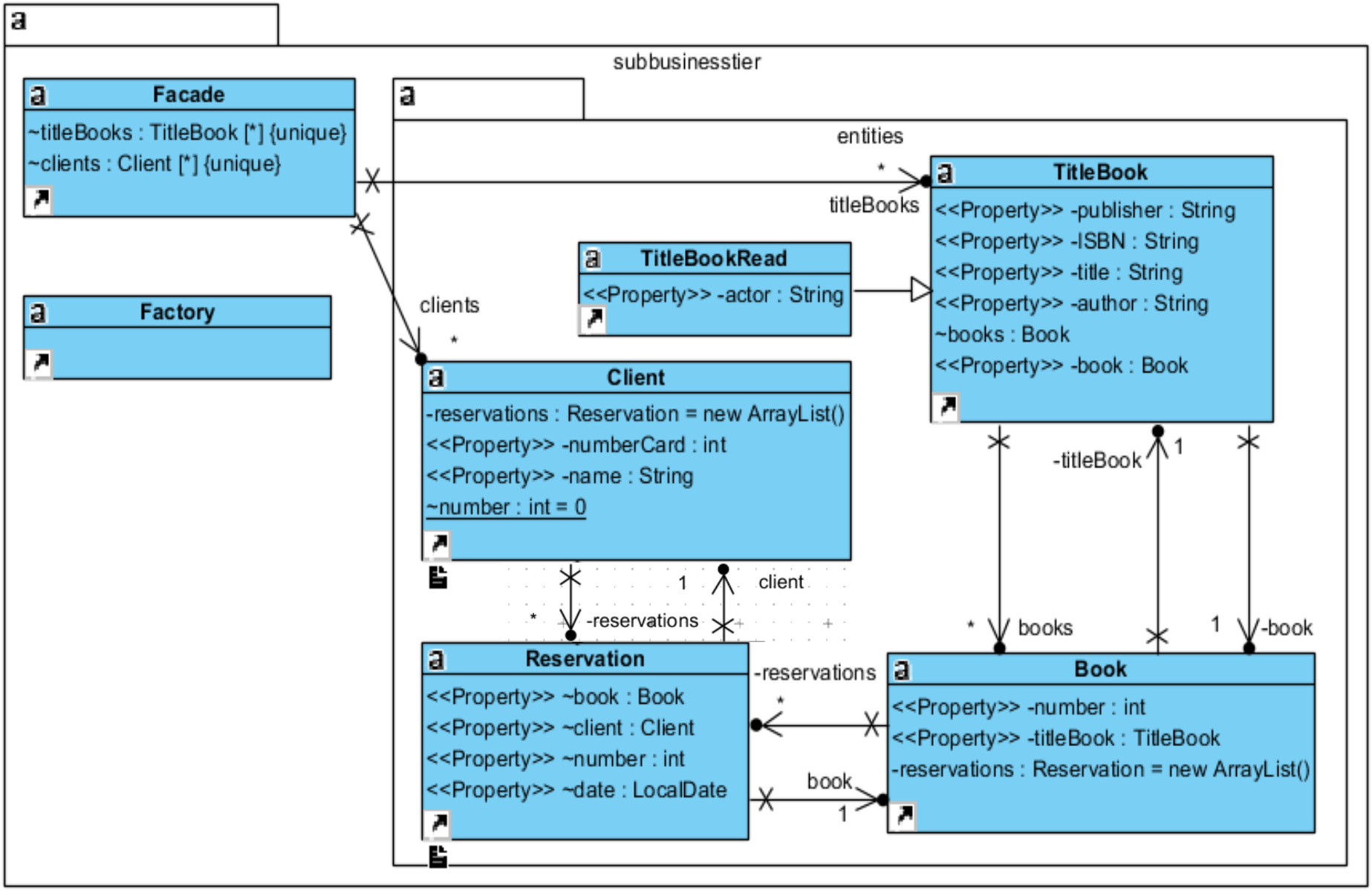
Drawing activity diagrams

(https://www.visual-paradigm.com/support/documents/vpuserguide/94/2580/6713_drawingactiv.html)

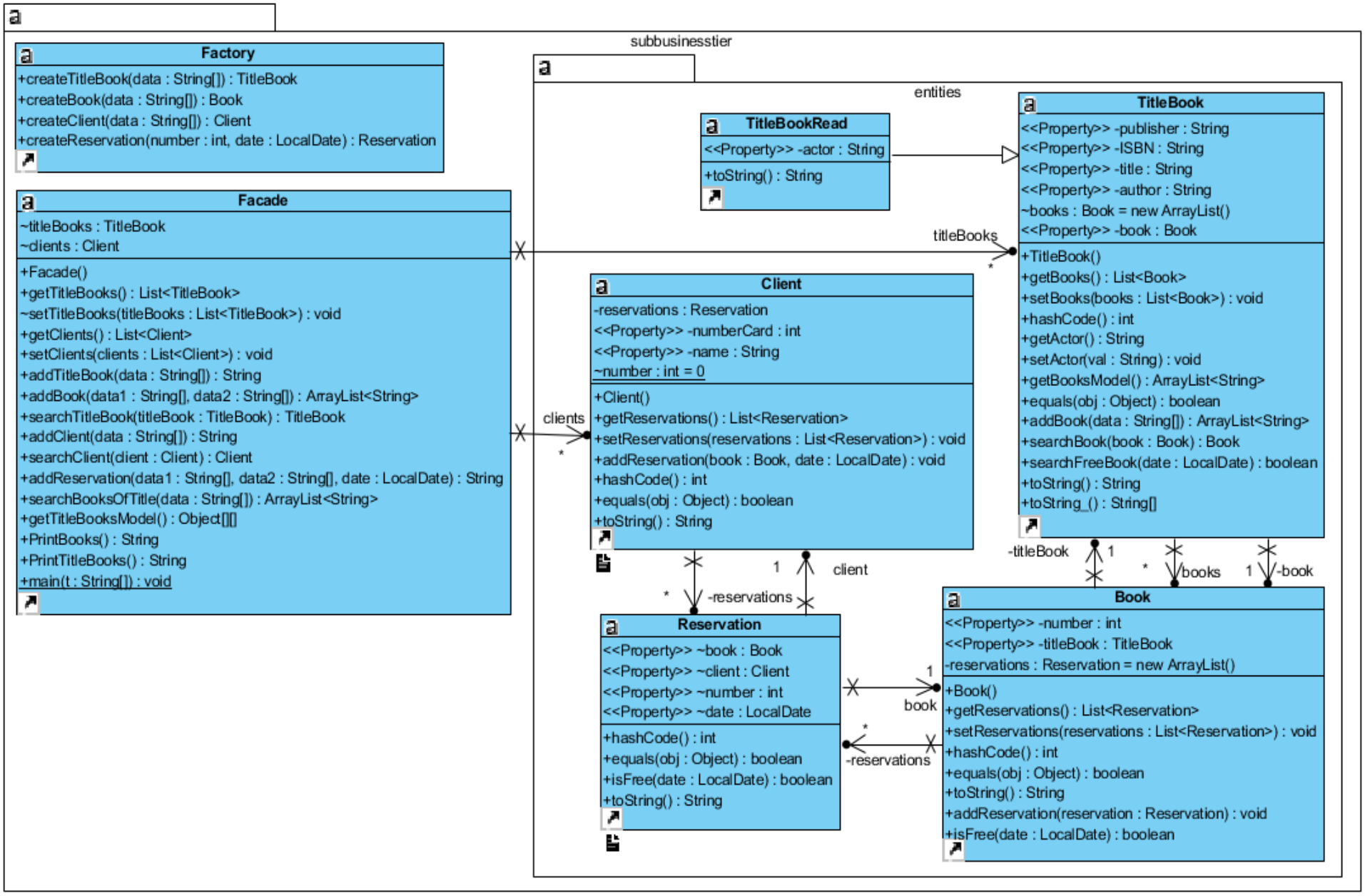
Diagram przypadków użycia (wykład 4 część 1, przykład 3) – wybrany fragment



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 subbusinesstier;
import java.time.LocalDate;
import java.time.Month;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import subbusinesstier.entities.Client;
import subbusinesstier.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) {}
public Client searchClient(Client client) {}
public String addClient(String data[]) {}
public String addTitleBook(String data[]) {}
public ArrayList<String> addBook(String data1[], String data2[]) {}
public String addReservation(String data1[], String data2[], LocalDate date) {}
//pomocnicze metody
   public ArrayList<String> searchBooksOfTitle(String data[])   {}
   public Object[][] getTitleBooksModel()                       {}
   public String PrintBooks()                                   {}
   public String PrintTitleBooks()                              {}
   public static void main(String t[])                          {}
}
```

PU Operacje_Na_Ksiazkach

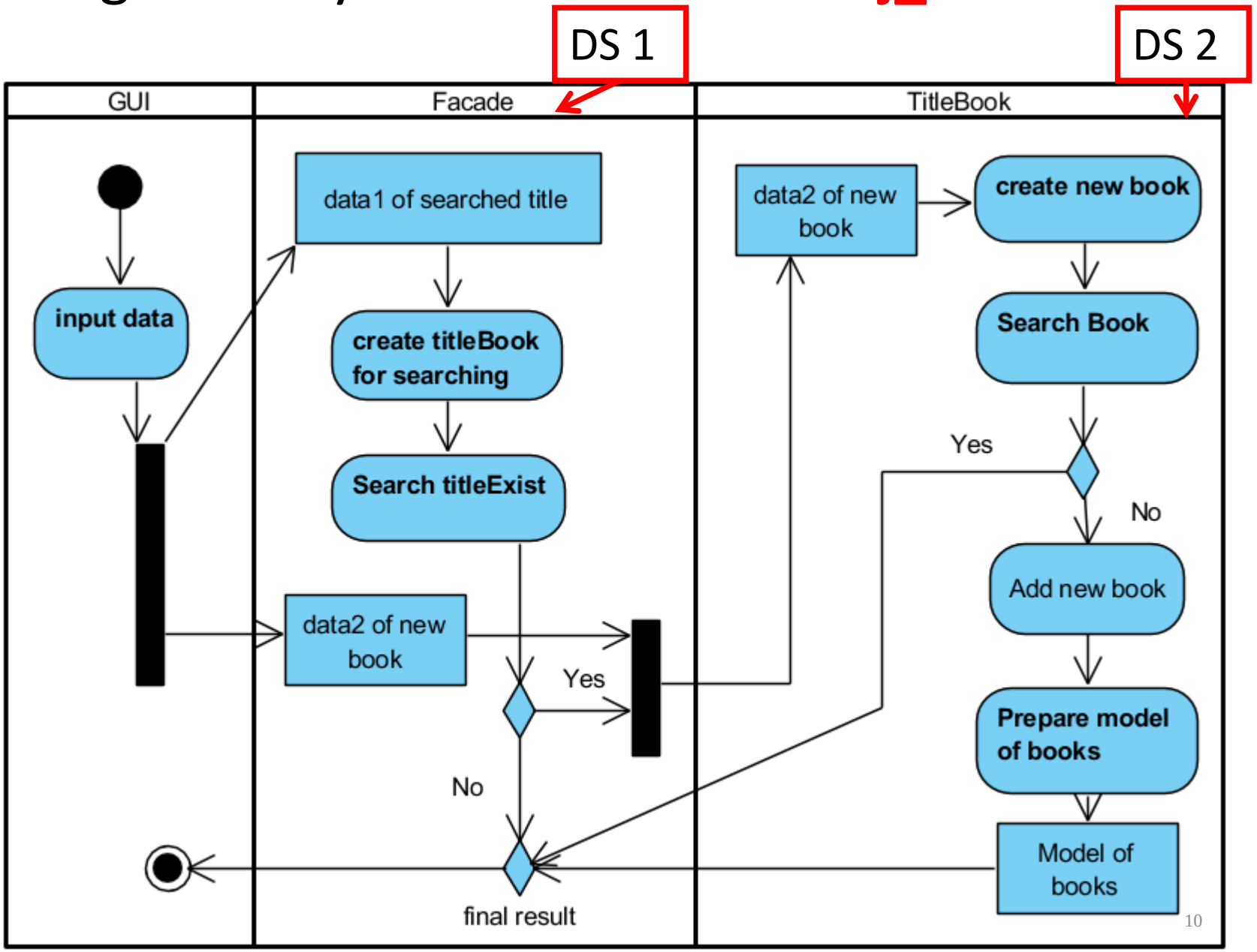
PU Rejestracja_Klienta

PU Dodaj_Tytul_Ksiazki

PU Dodaj_Ksiazke

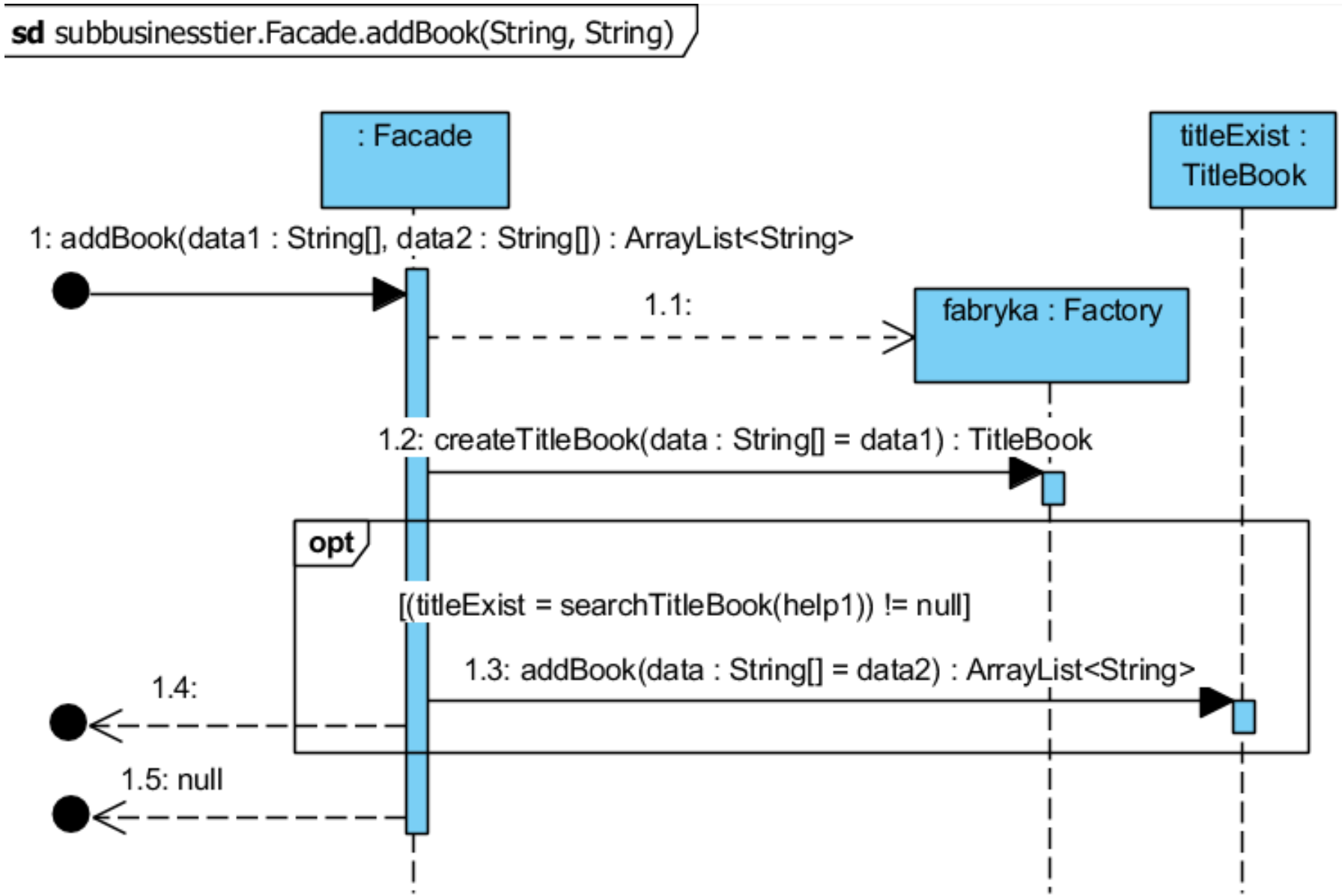
PU Rezerwacja

1. Diagram aktywności dla **PU Dodaj_ksiazke**



(DS1) Wstawianie nowej książki o podanym tytule – 1-y etap

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

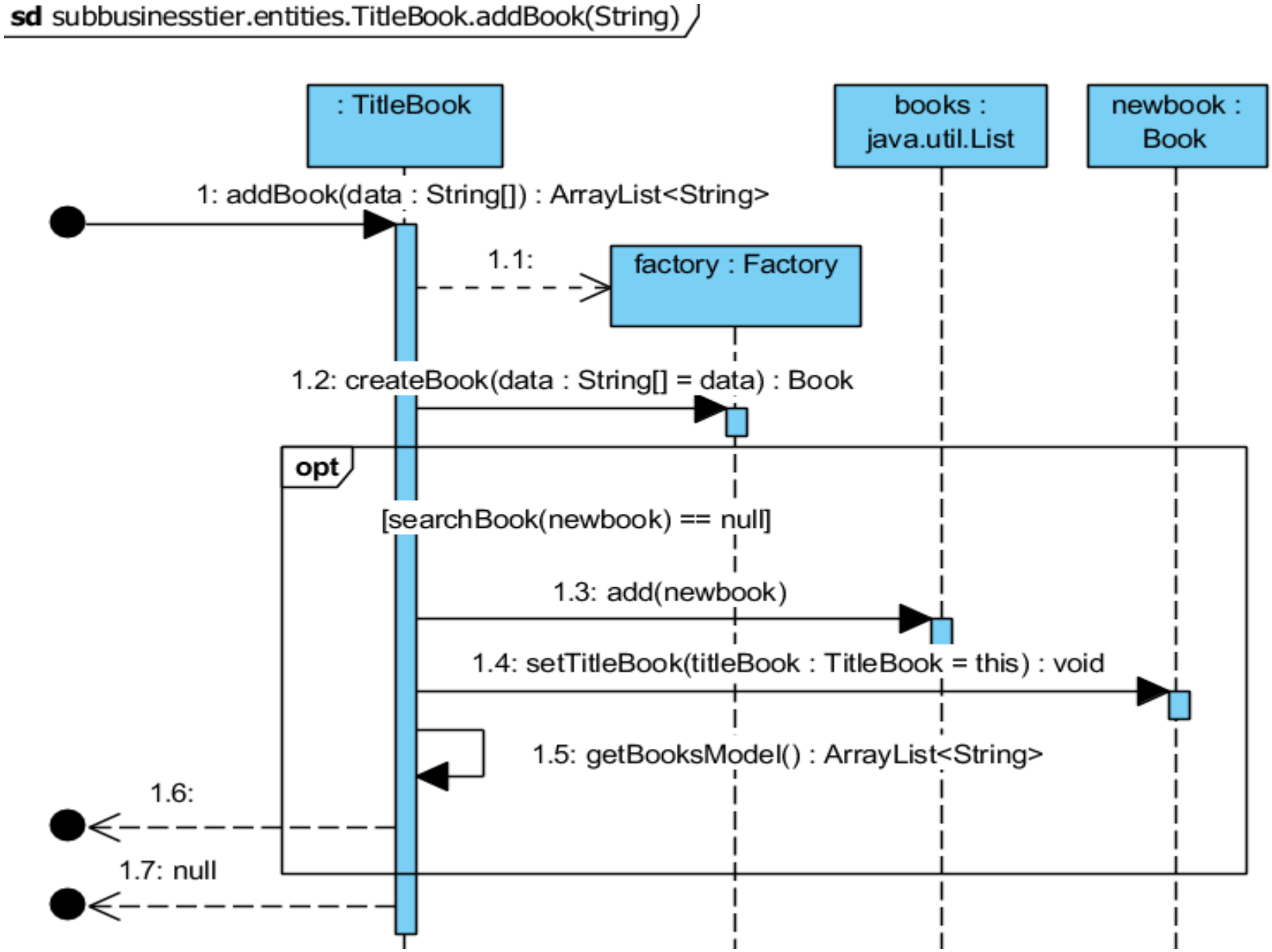


```
//class Facade
  List<TitleBook> titleBooks;
  List<Client> clients;
  public Facade() {
     titleBooks = new ArrayList<>();
     clients = new ArrayList();
  }

public ArrayList<String> addBook(String data1[], String data2[]) {
     TitleBook help1, titleExist;
     Factory fabryka = new Factory();
     help1 = fabryka.createTitleBook(data1);
     if ((titleExist = searchTitleBook(help1)) != null) {
         return titleExist.addBook(data2);
        }
     return null;
  }
```

(DS2) Wstawianie nowej książki o podanym tytule – 2-i etap

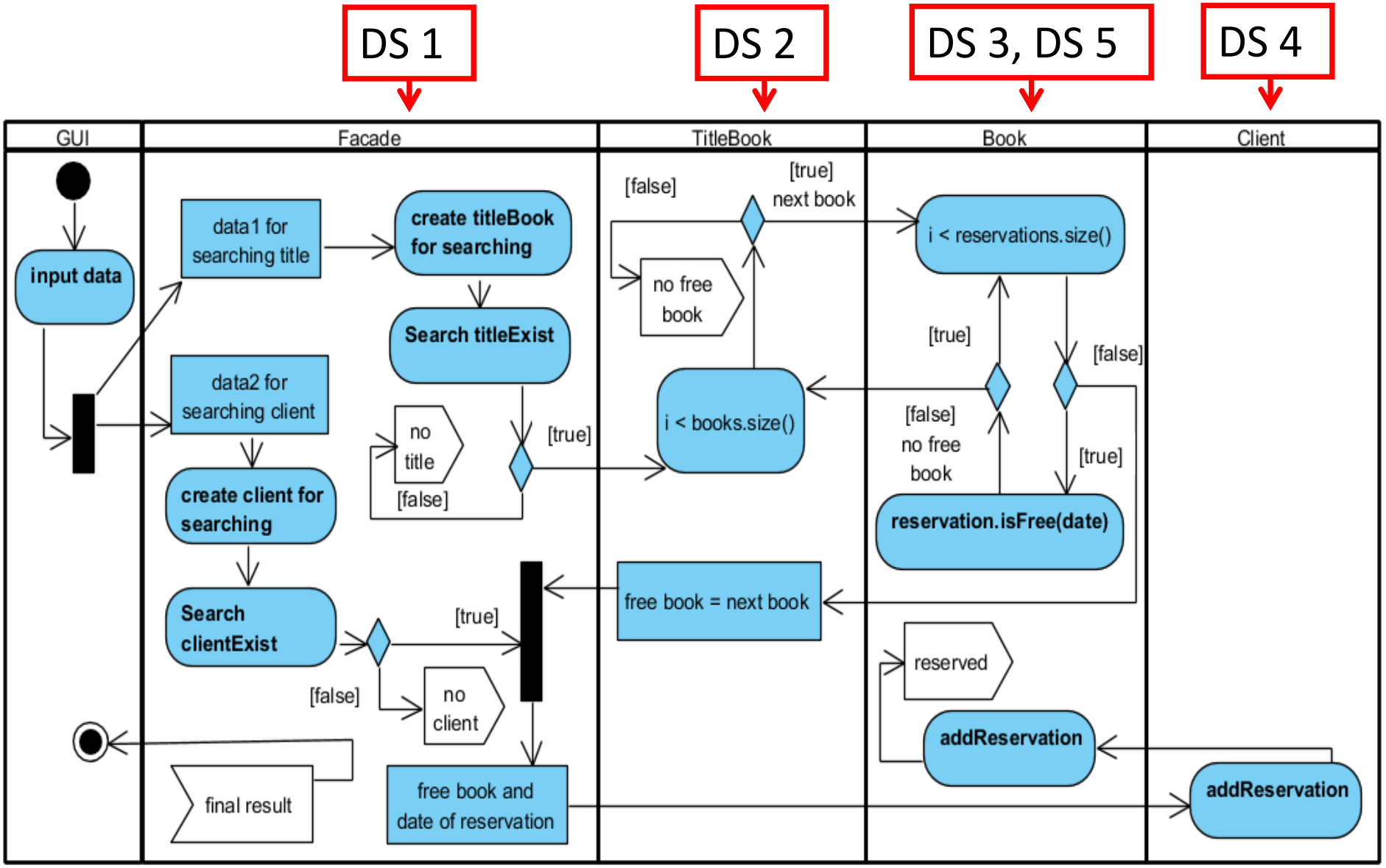
public ArrayList<String> addBook(String data[])



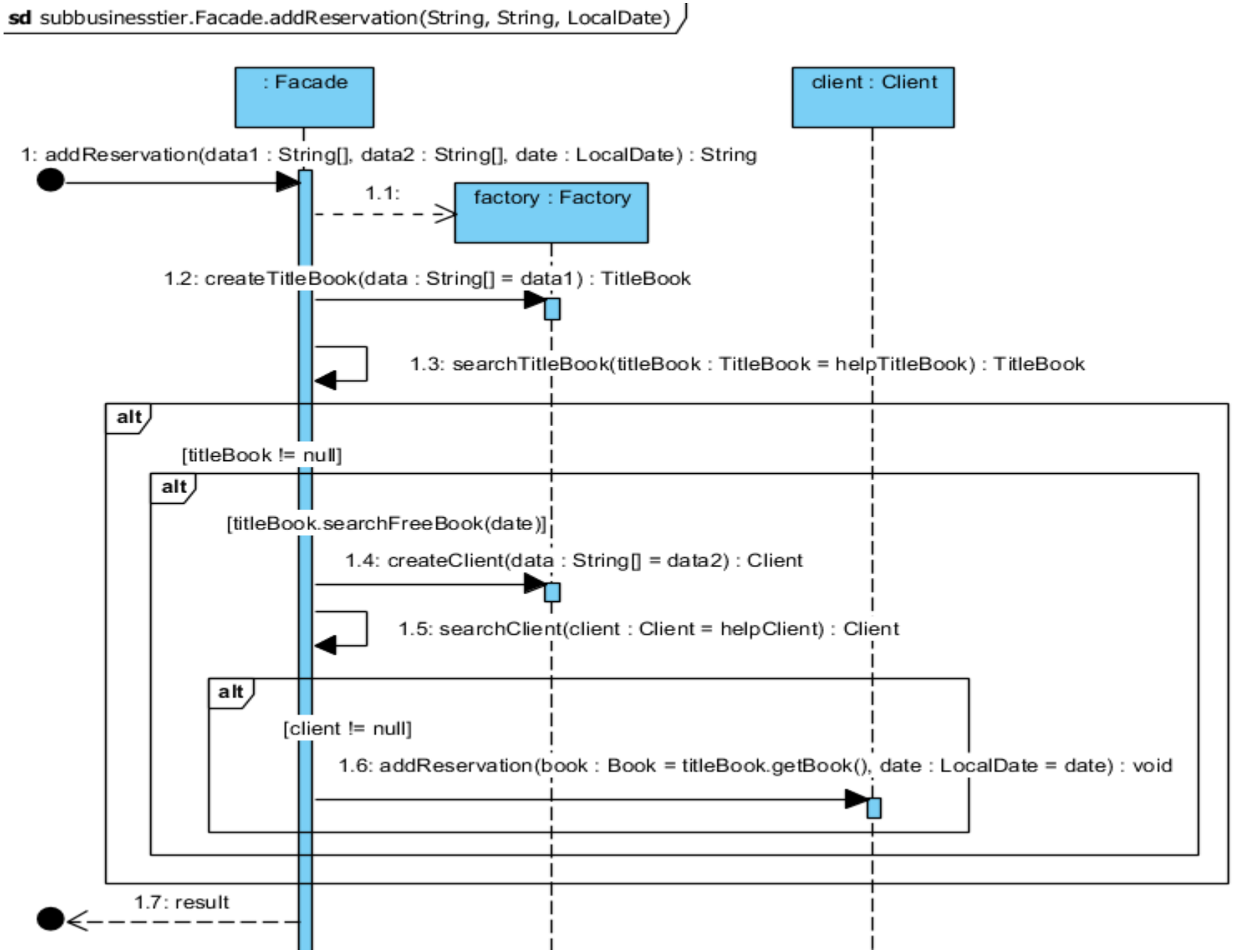
```
//class TitleBook
  List<Book> books;
  public TitleBook() {
     books = new ArrayList();
  }

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;
  }
```

2. Diagram aktywności dla **PU Rezerwacja**



(DS1) Rezerwacja książki: public String addReservation(String data1[], String data2[], LocalDate date)

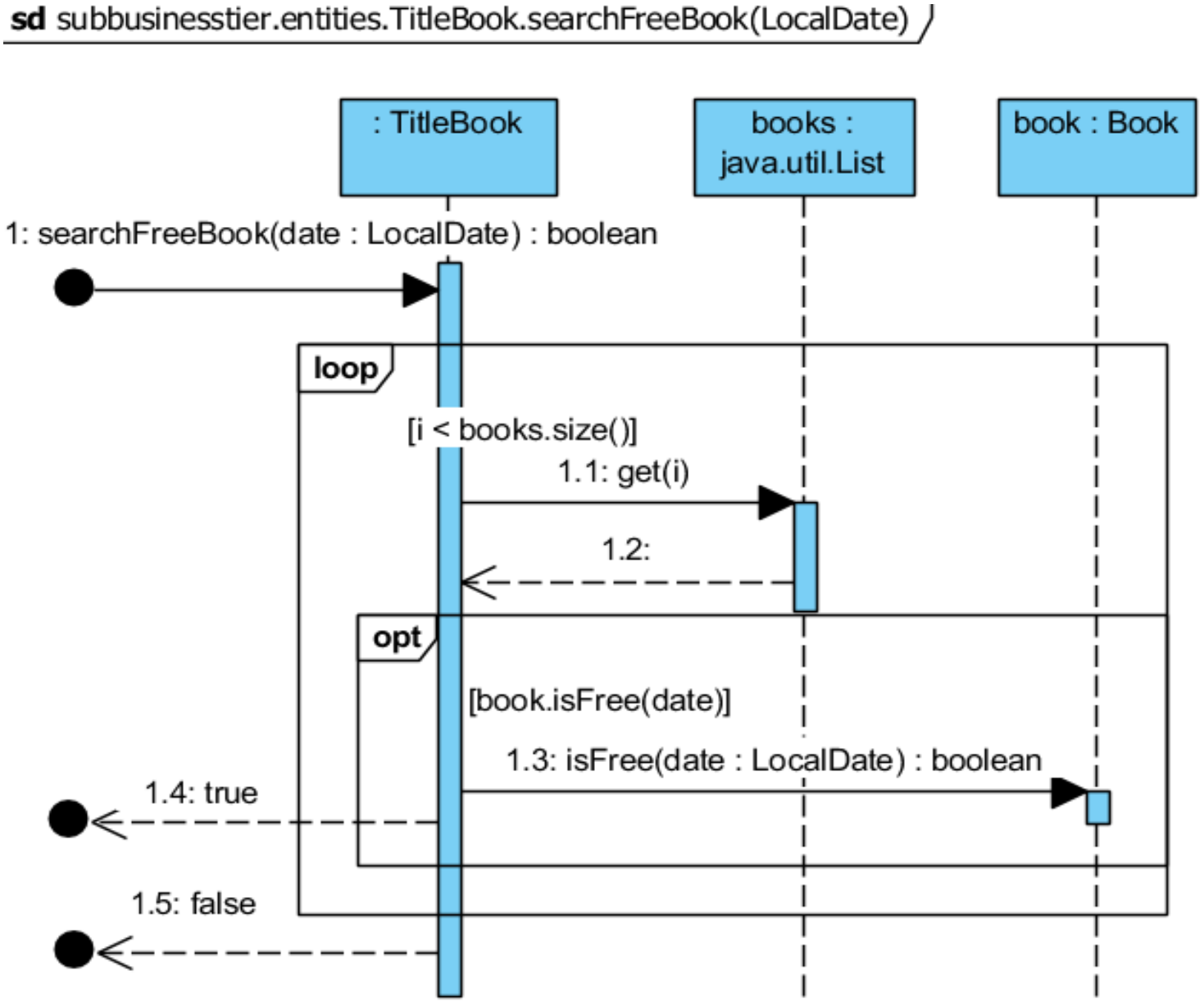


Identyfikacja torów na diagramie aktywności dla procesu Rezerwacja

```
                                        //class Facade
public String addReservation(String data1[], String data2[], LocalDate date) {
    String result;
    Factory factory = new Factory();
    TitleBook helpTitleBook = factory.createTitleBook(data1), titleBook;
    titleBook = this.searchTitleBook(helpTitleBook);
    if (titleBook != null)
       if (titleBook.searchFreeBook(date)) {          //book.isFree(date)
          Client helpClient = factory.createClient(data2), client;
           client = this.searchClient(helpClient);
          if (client != null) {
             client.addReservation(titleBook.getBook(), date);
             result = "reserved";
          } else  result = "no such a client";
       } else  result = "no free book";
     else  result = "no such a title";
    return result;
 }
```

(DS2) Wyszukiwanie wolnej książki do rezerwacji

public boolean searchFreeBook(LocalDate date)

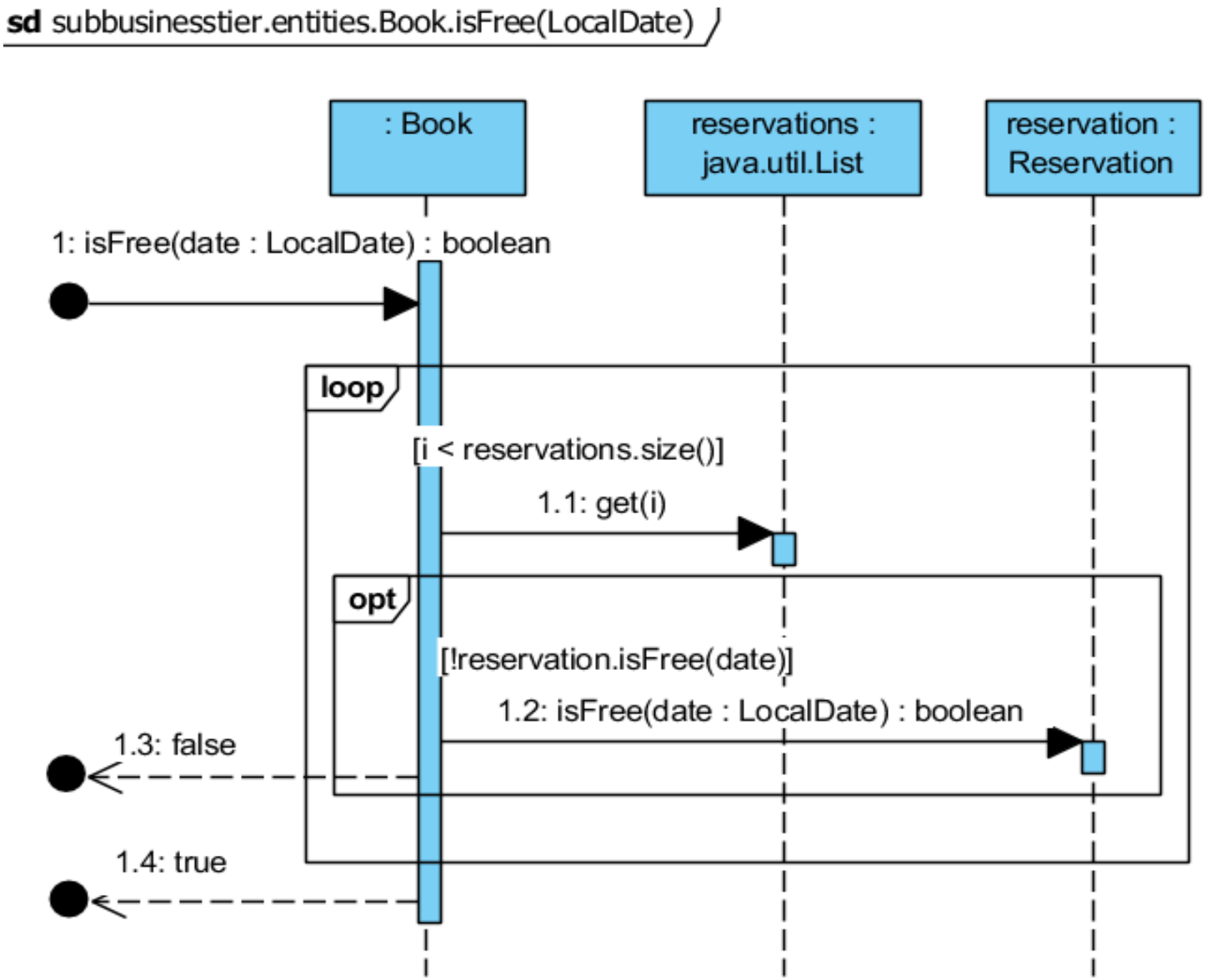


```
//class TitleBook
   List<Book> books;
   public TitleBook() {
      books = new ArrayList();
   }
  private Book book;   //atrybut book przechowuje obiekt typu
                       //Book wyszukany do rezerwacji

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;
   }
```

(DS3) Sprawdzanie przez książkę, czy ma wolny termin rezerwacji

public boolean isFree(LocalDate date)

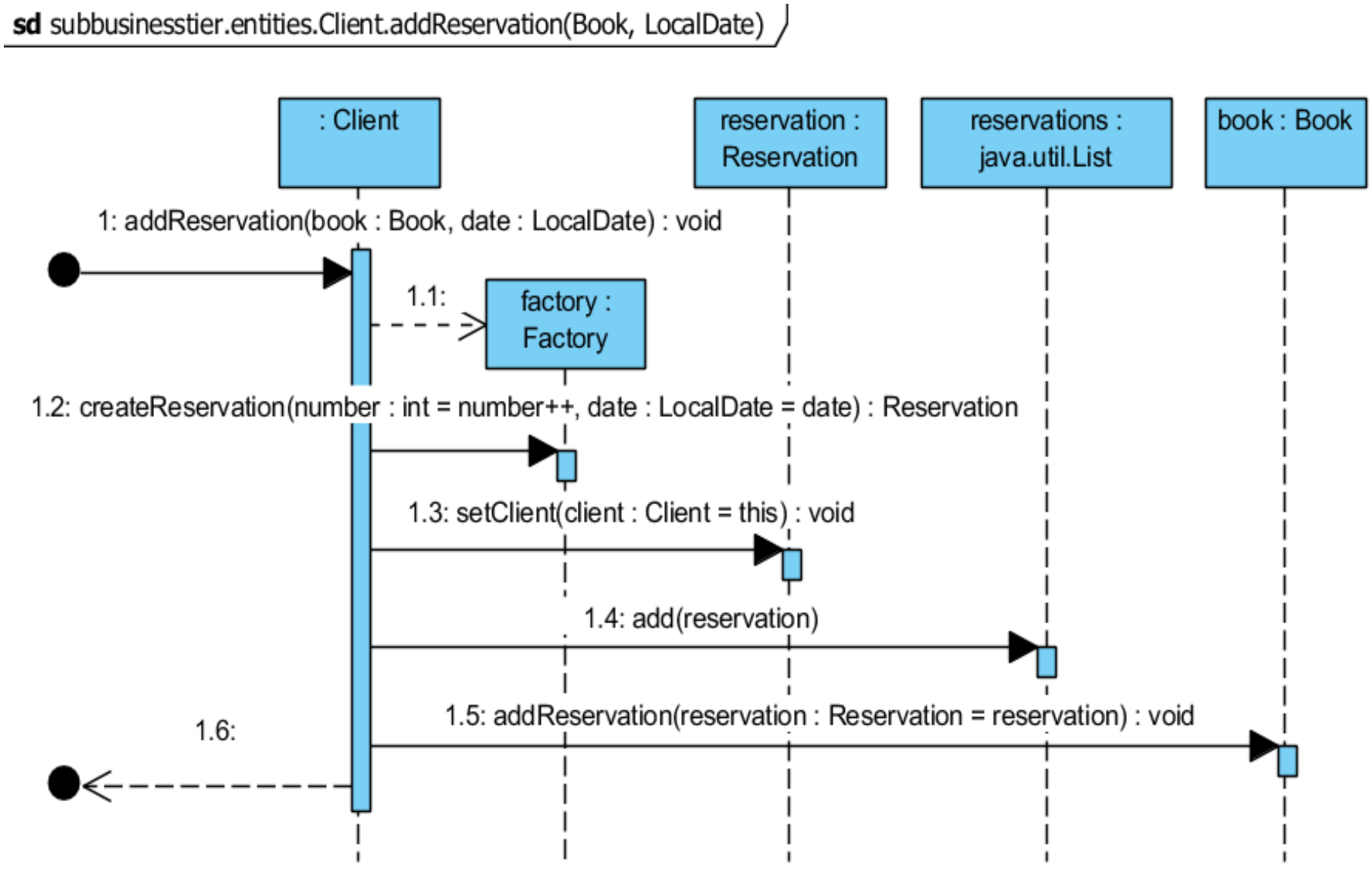


```
                                        //class Book
 private List<Reservation> reservations;
 public Book() {
       reservations = new ArrayList();
   }

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;
   }
```

(DS4) Wykonanie rezerwacji przez obiekt typu Client – 1-y etap

public void addReservation(Book book, LocalDate date)

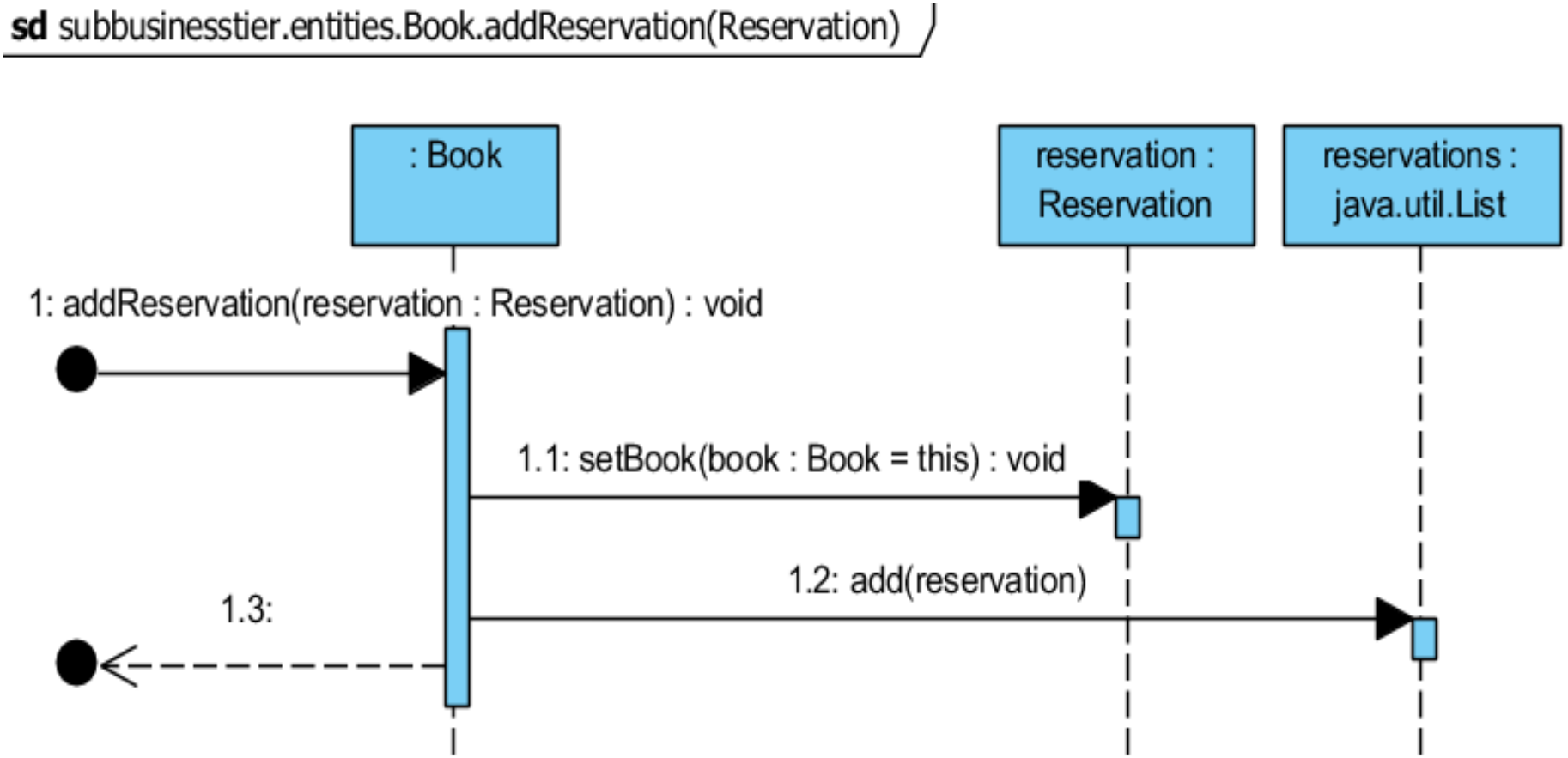


```
                                    //class Client
private List<Reservation> reservations;
public Client() {
      reservations=new ArrayList();
  }

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);
  }
```

(DS5) Wykonanie rezerwacji przez obiekt typu Book – 2-i etap

public void addReservation(Reservation reservation)



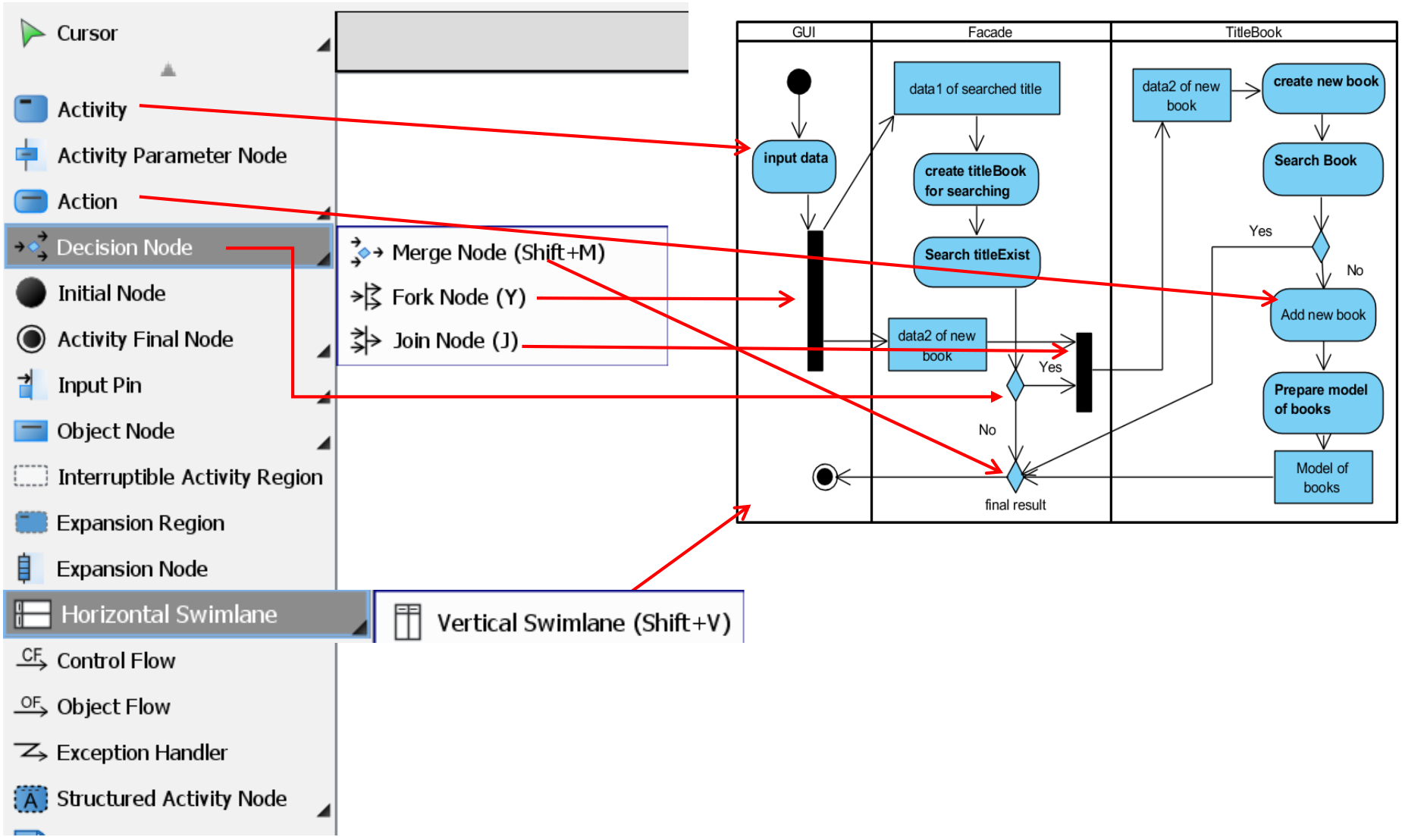
```
// class Book

private List<Reservation> reservations;
 public Book() {
      reservations = new ArrayList();
   }

 public void addReservation(Reservation reservation) {
      reservation.setBook(this);
      reservations.add(reservation);
   }
```

Dodatek – informacja tworzeniu diagramu aktywności

Informacja tworzeniu diagramu aktywności – część 1



Informacja tworzeniu diagramu aktywności – część 2