Wybrane diagramy opisu systemu w języku UML Wykład6

Zofia Kruczkiewicz

Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności. Diagramy stanów

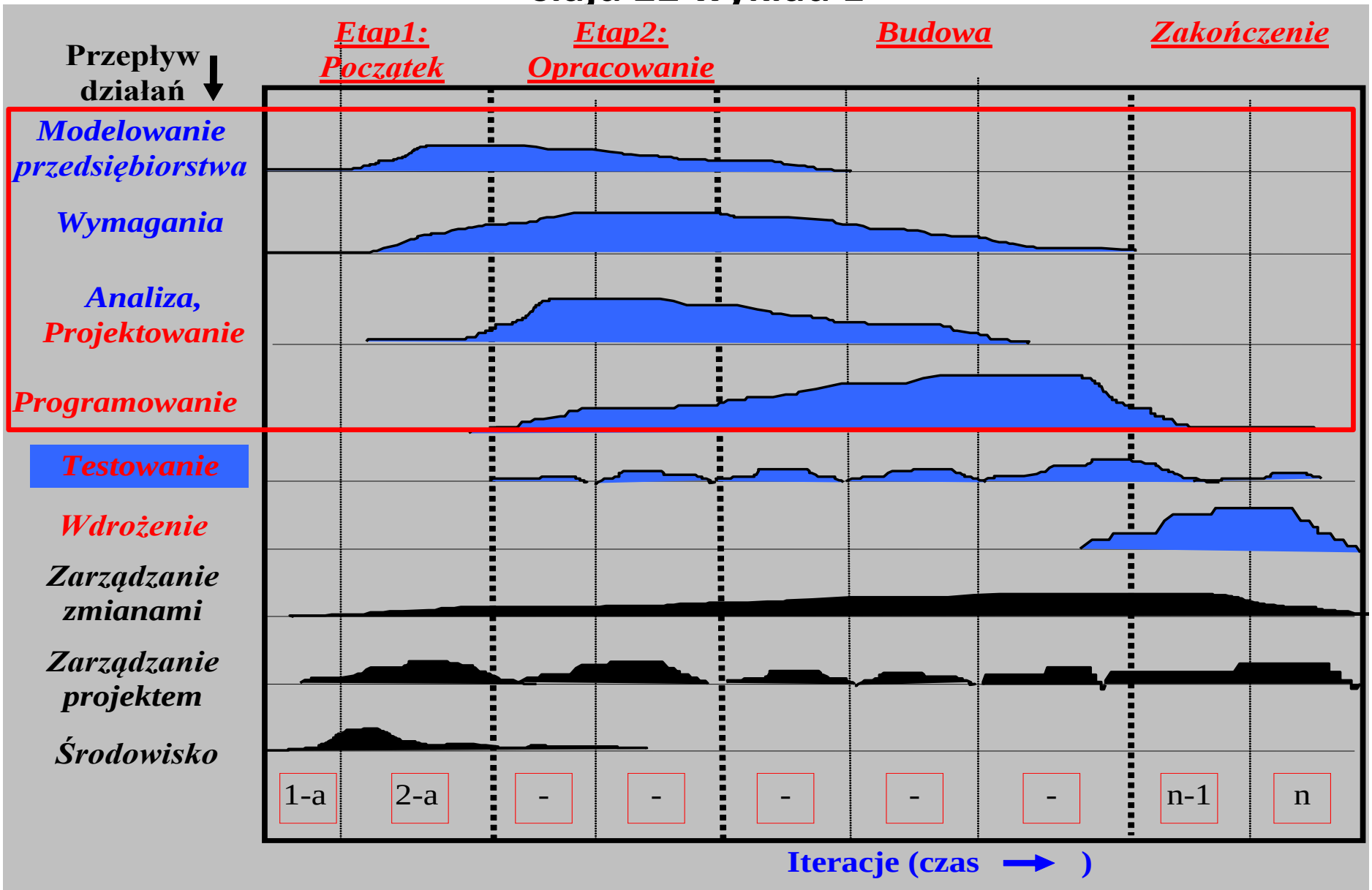
1. Wprowadzenie
2. Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności - porównanie
2. Diagramy stanów UML

   https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html
3. Przykłady diagramów stanu

Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności. Diagramy stanów

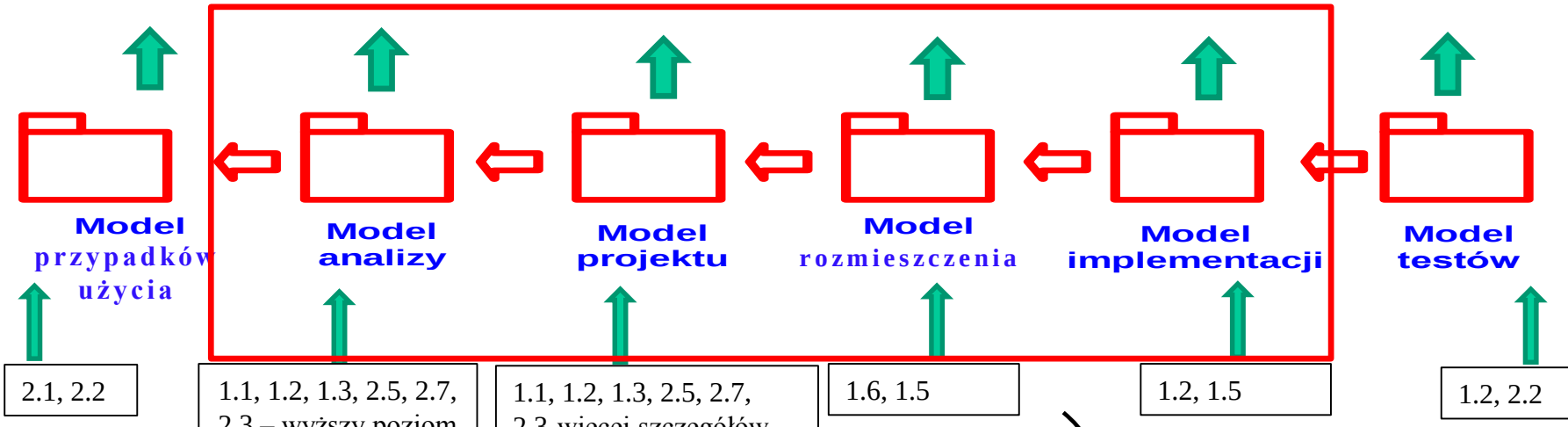
1. Wprowadzenie

Proces - zunifikowany iteracyjno- przyrostowy proces tworzenia oprogramowania – kiedy należy wykonać? [3LU]
- slajd 22 wykład 1



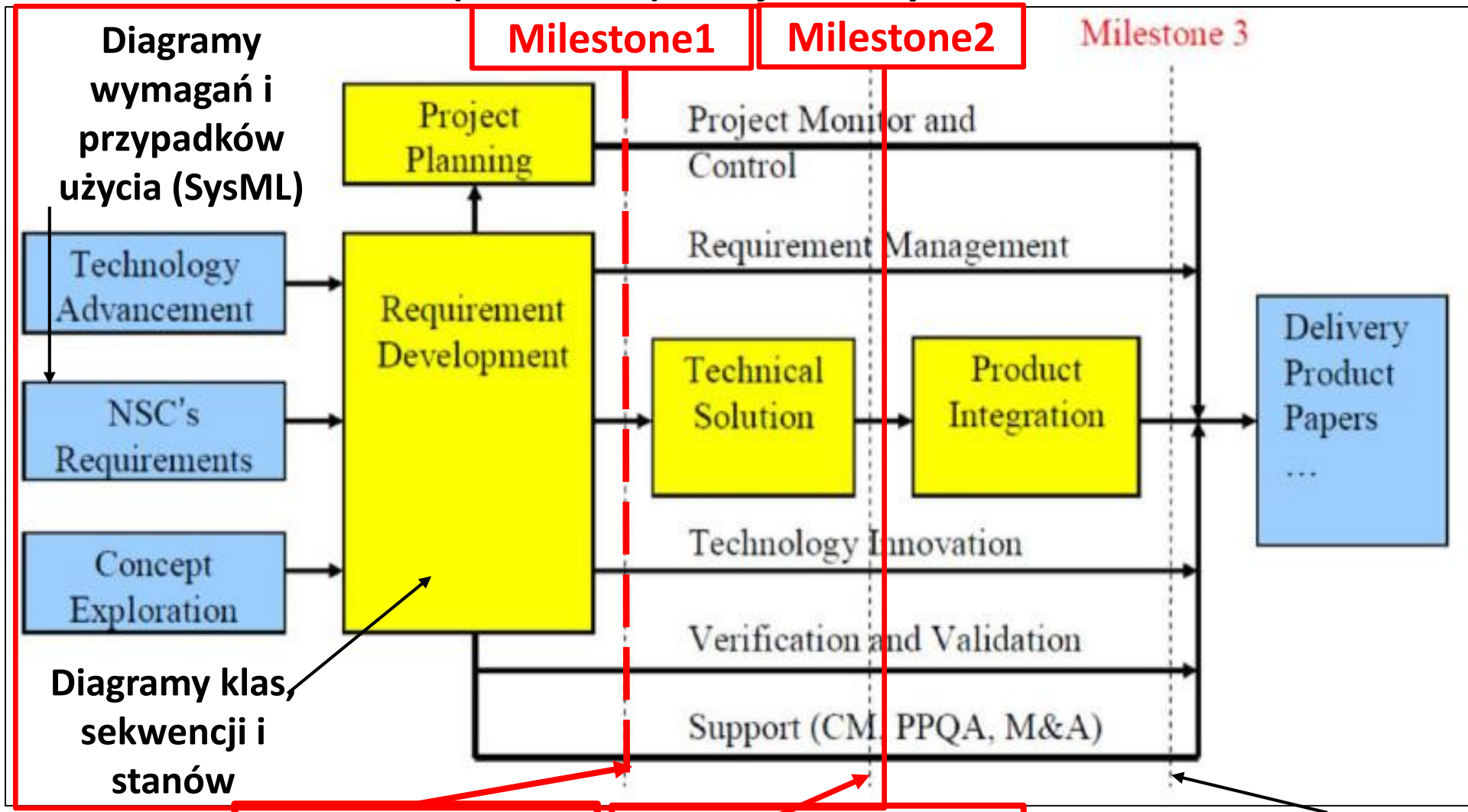
Produkt - diagramy UML – modele, proces - slajd 45, wykład 1

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



Numery diagramów UML: slajdy 34 i 35

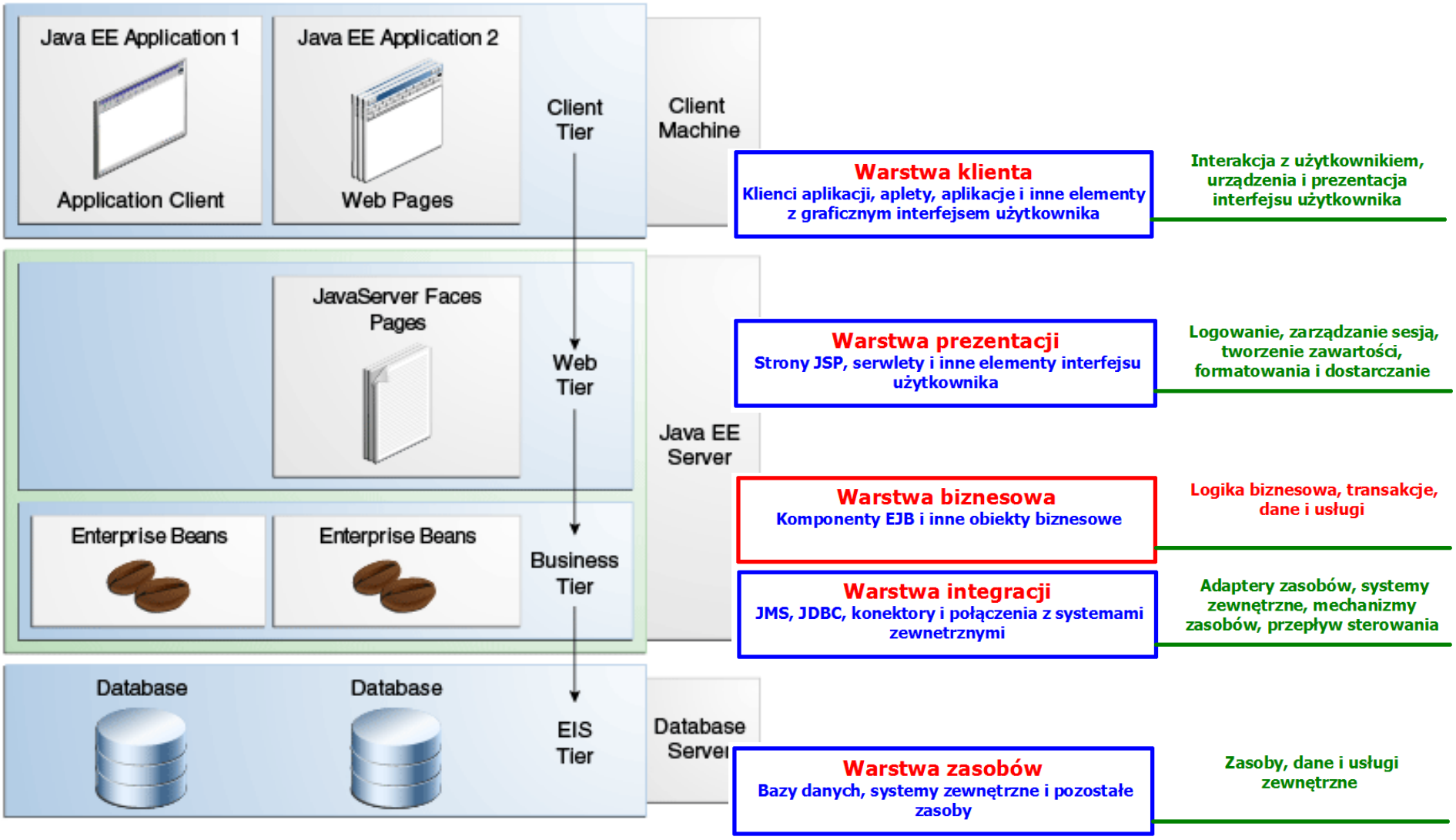
Cykl życia tworzenia oprogramowania w dziedzinie medycyny nuklearnej: Light-Weight Capability Maturity Model Integration (LW-CMMI) – slajd 59, wykład 1



Produkt – oprogramowanie na platformie Java EE

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

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

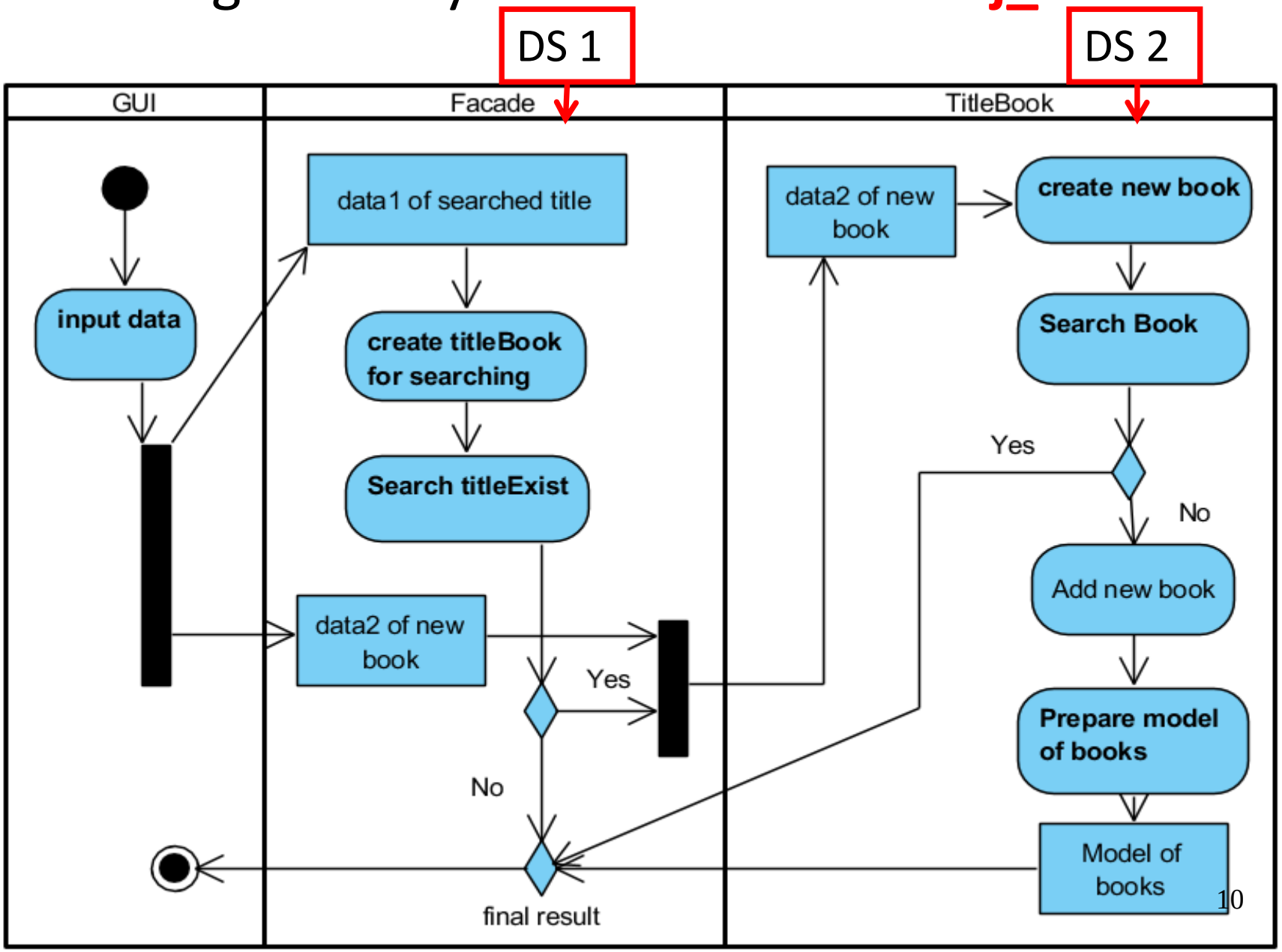


Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności. Diagramy stanów

1. Wprowadzenie

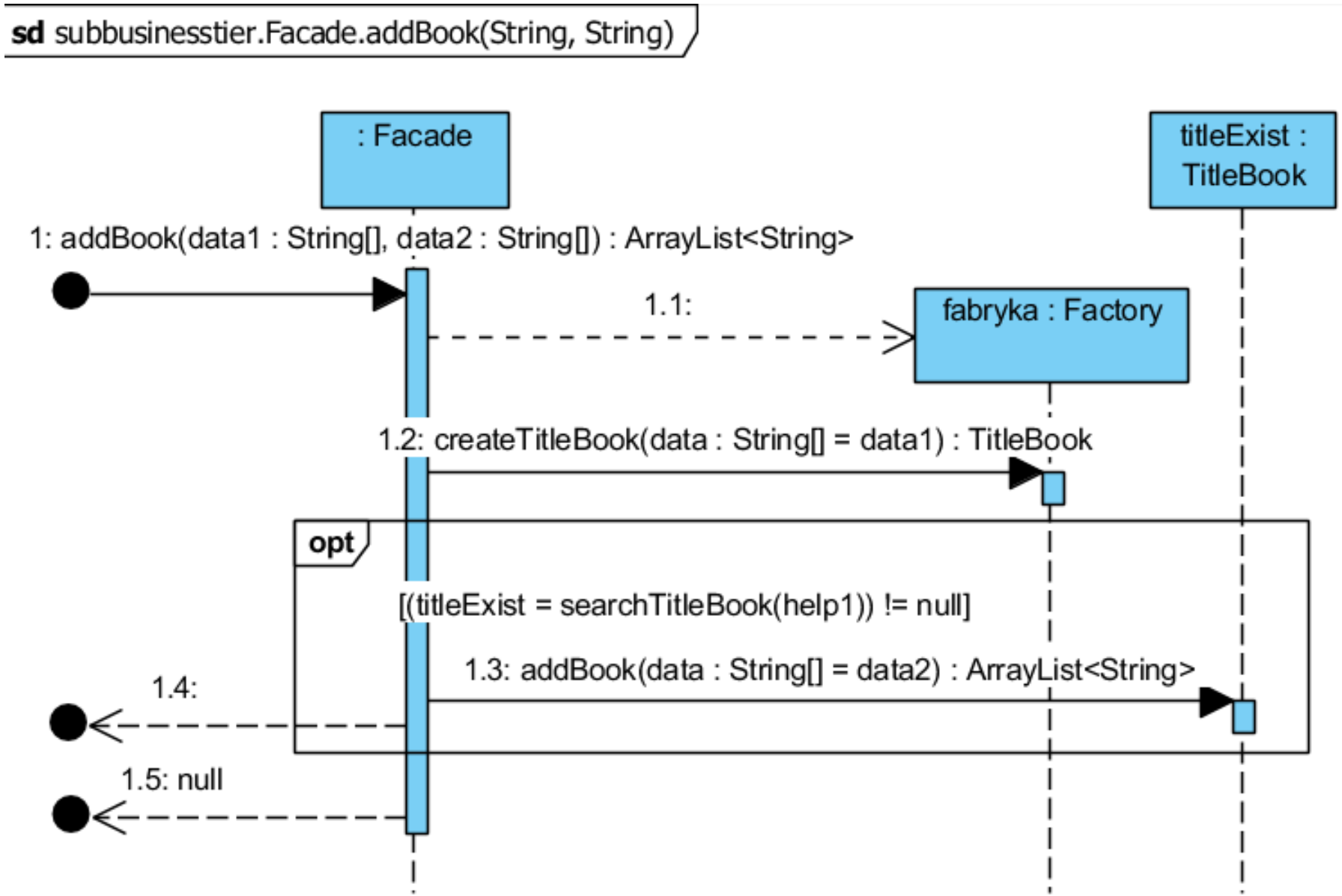
2. Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności - porównanie

1. Diagram aktywności dla **PU Dodaj_ksiazke**



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

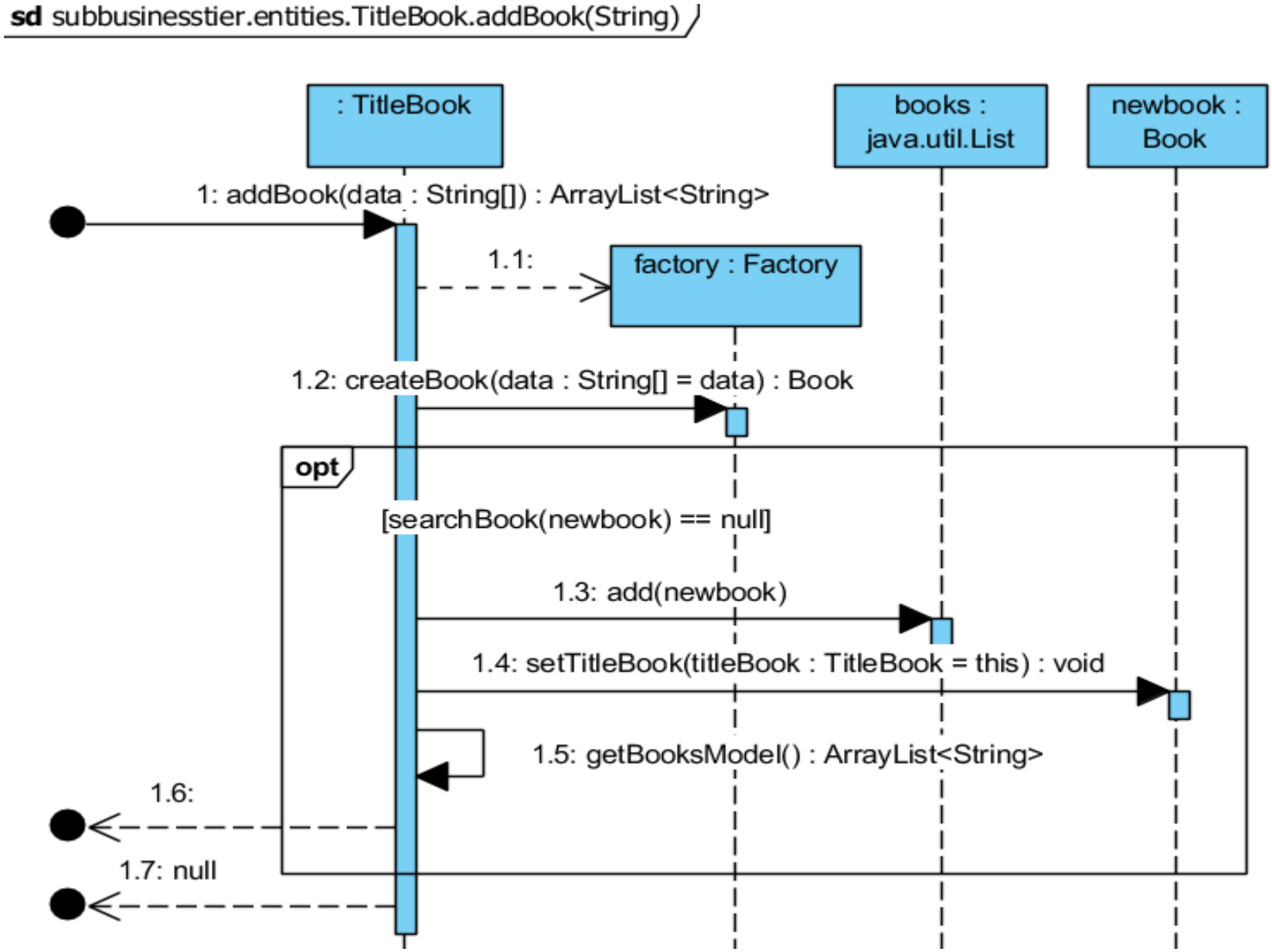
public ArrayList<String> addBook(String data1[], String data2[])



```
//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;
  }
```

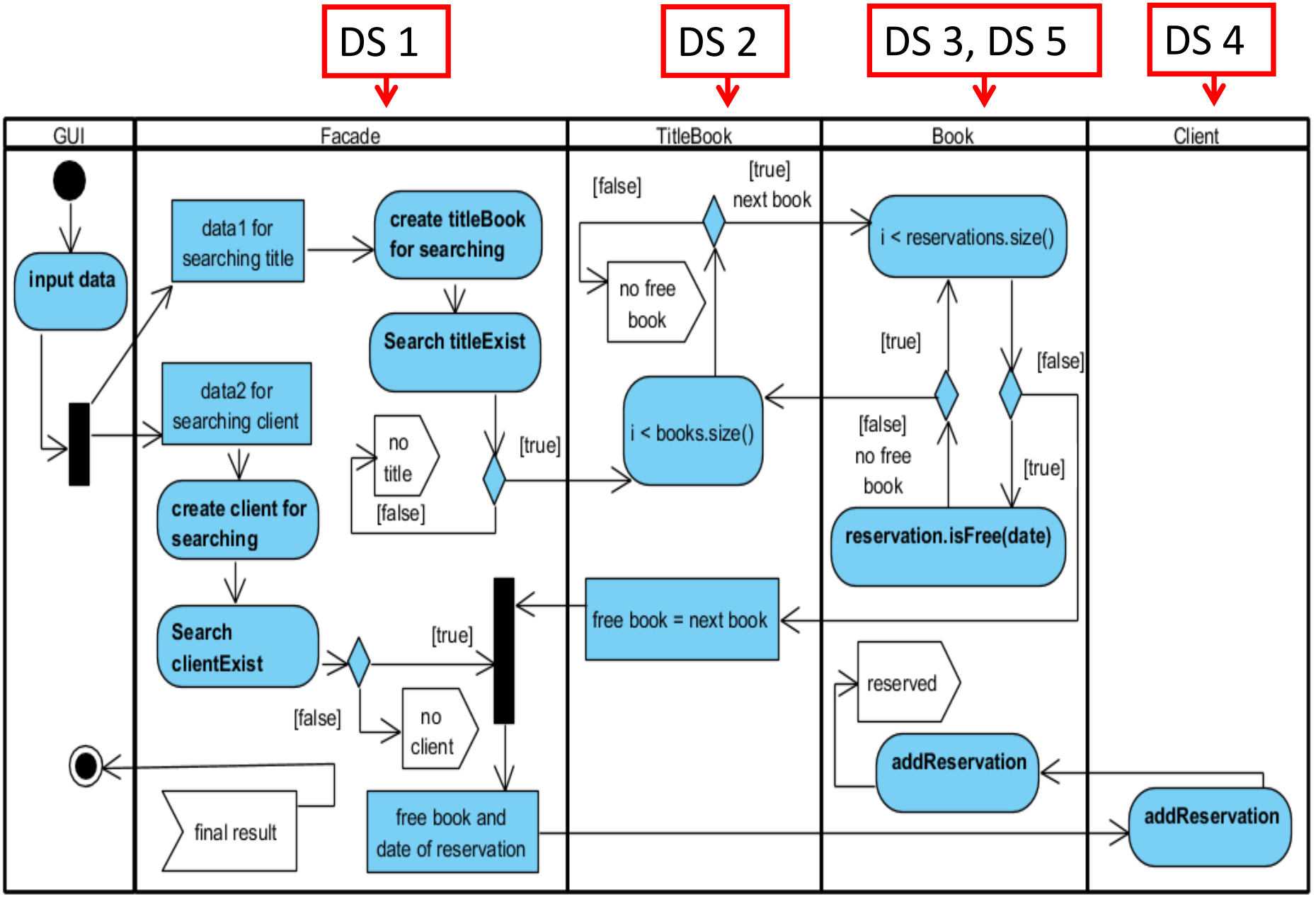
(DS2) Wstawianie nowej książki o podanym tytule – 2-i etap

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

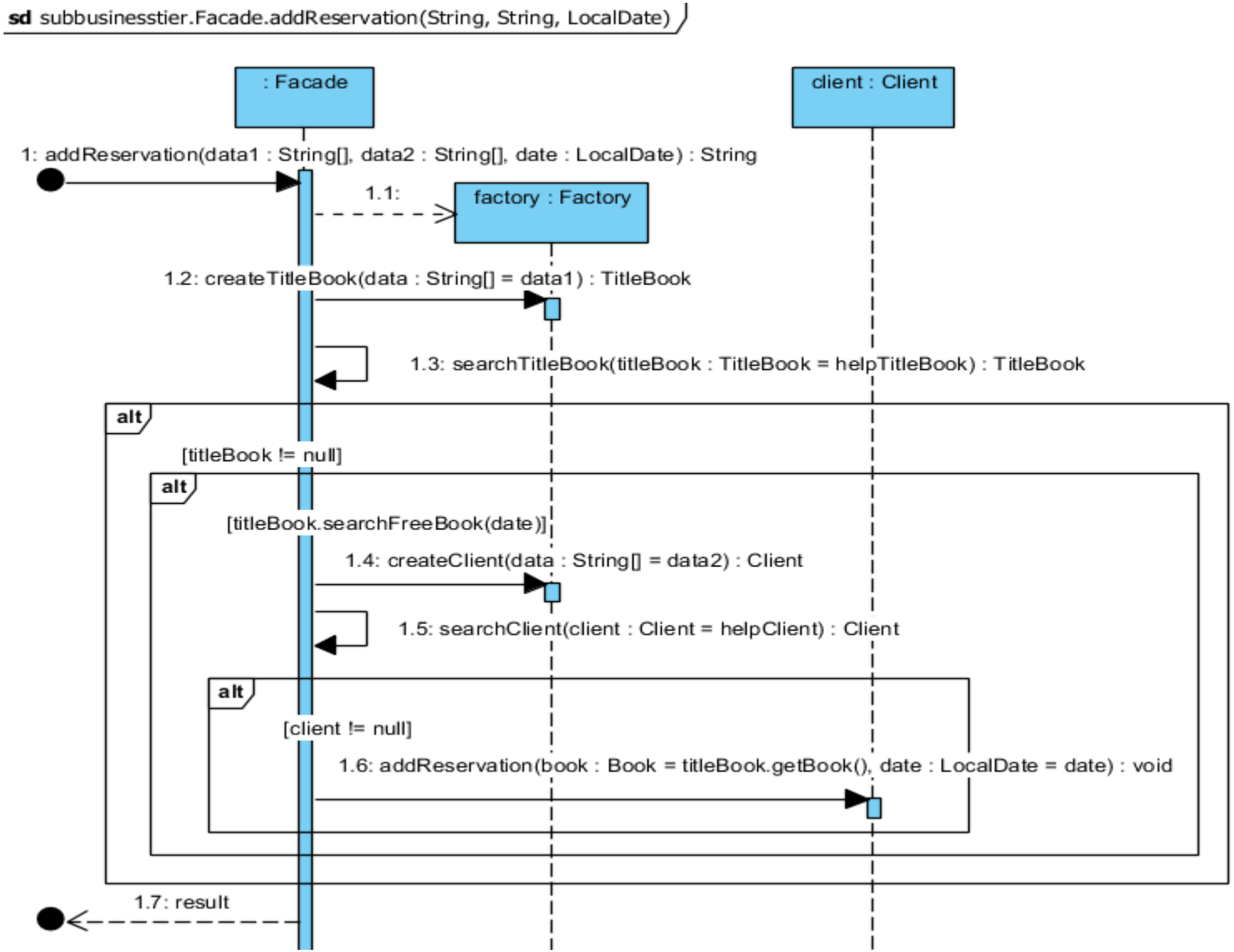


```
//class TitleBook
  List<Book> books;
  public TitleBook() {
     books = new ArrayList();
  }

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)

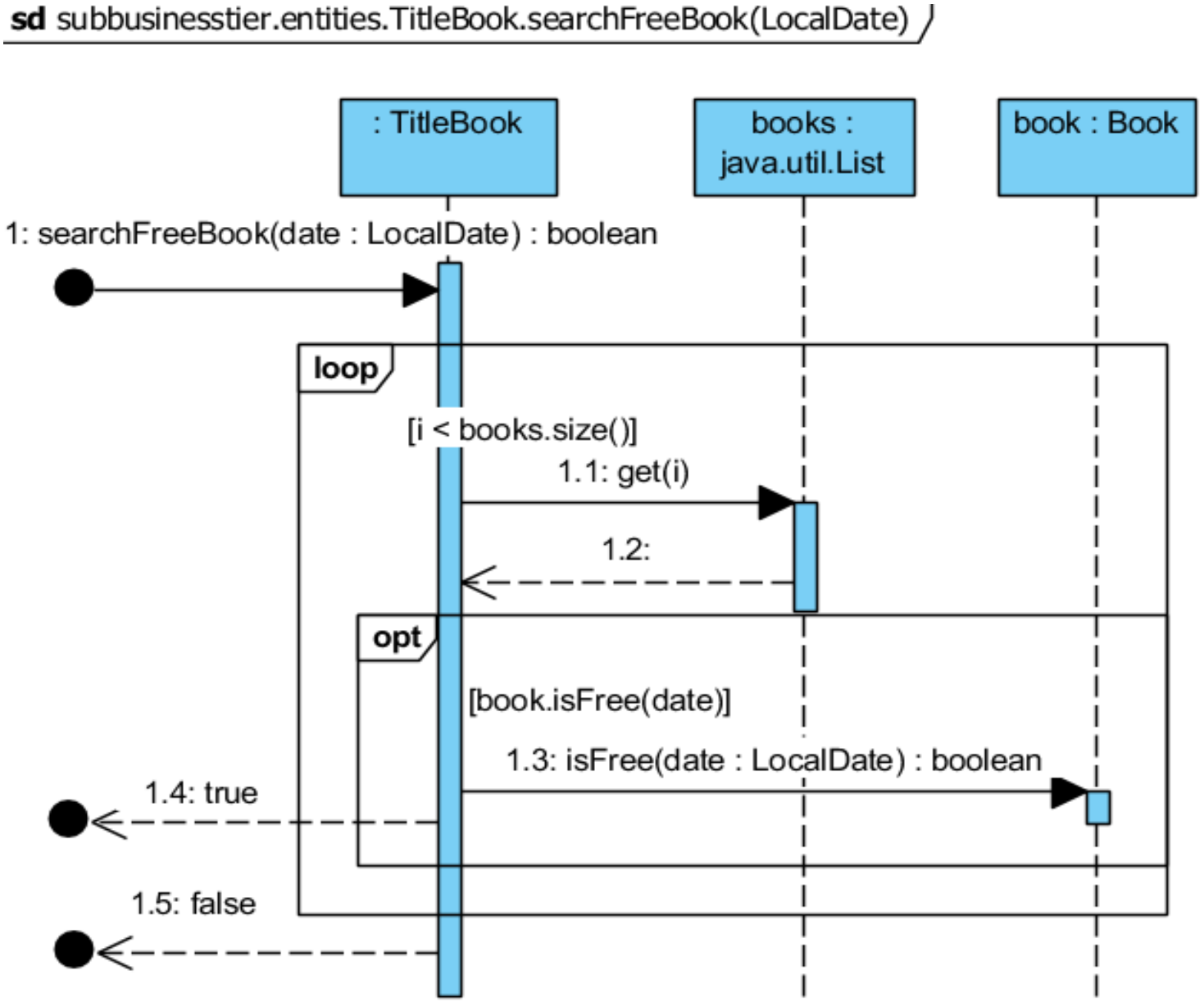


Identyfikacja torów na diagramie aktywności dla procesu Rezerwacja

```
                                        //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

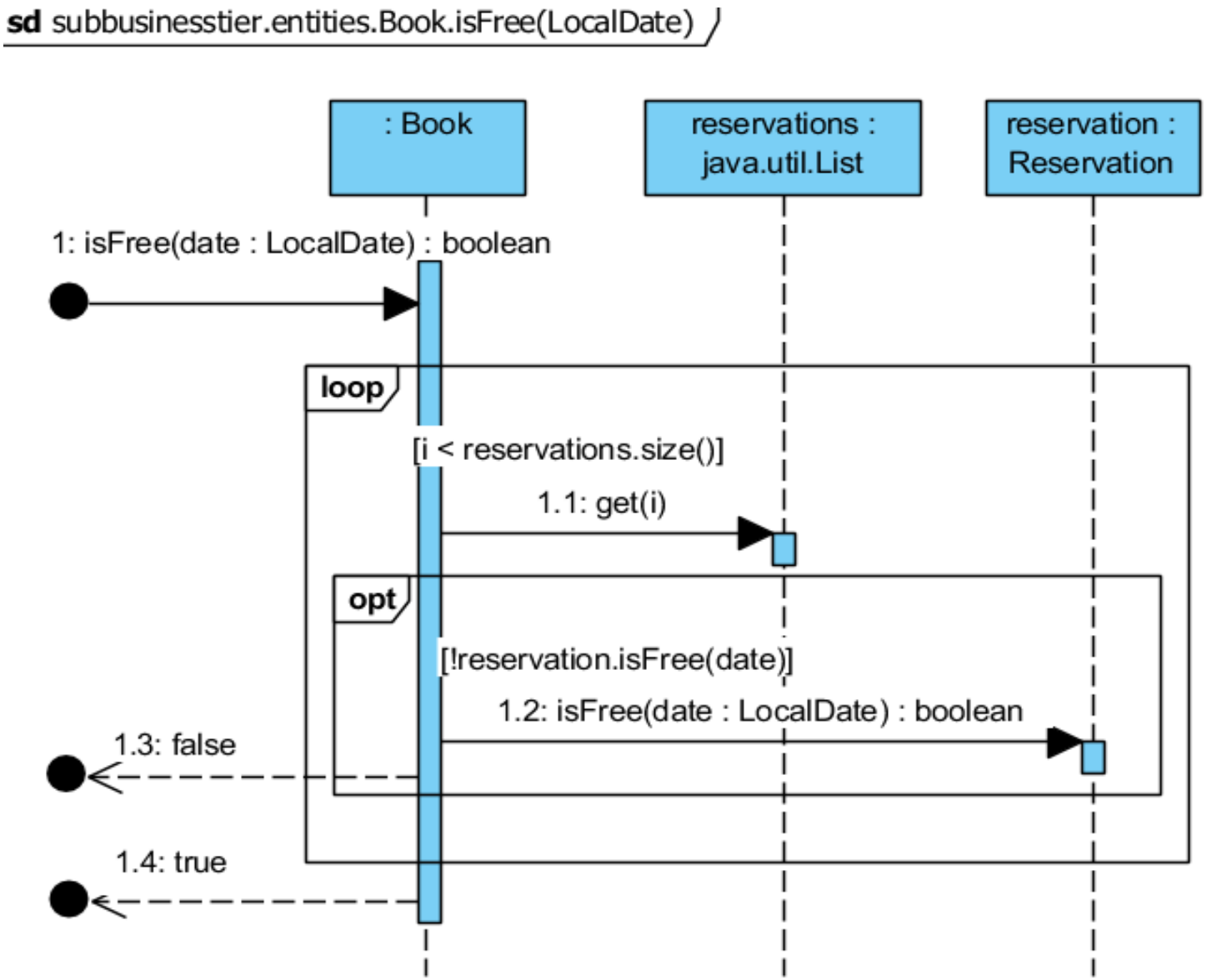
public boolean searchFreeBook(LocalDate date)



```
//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

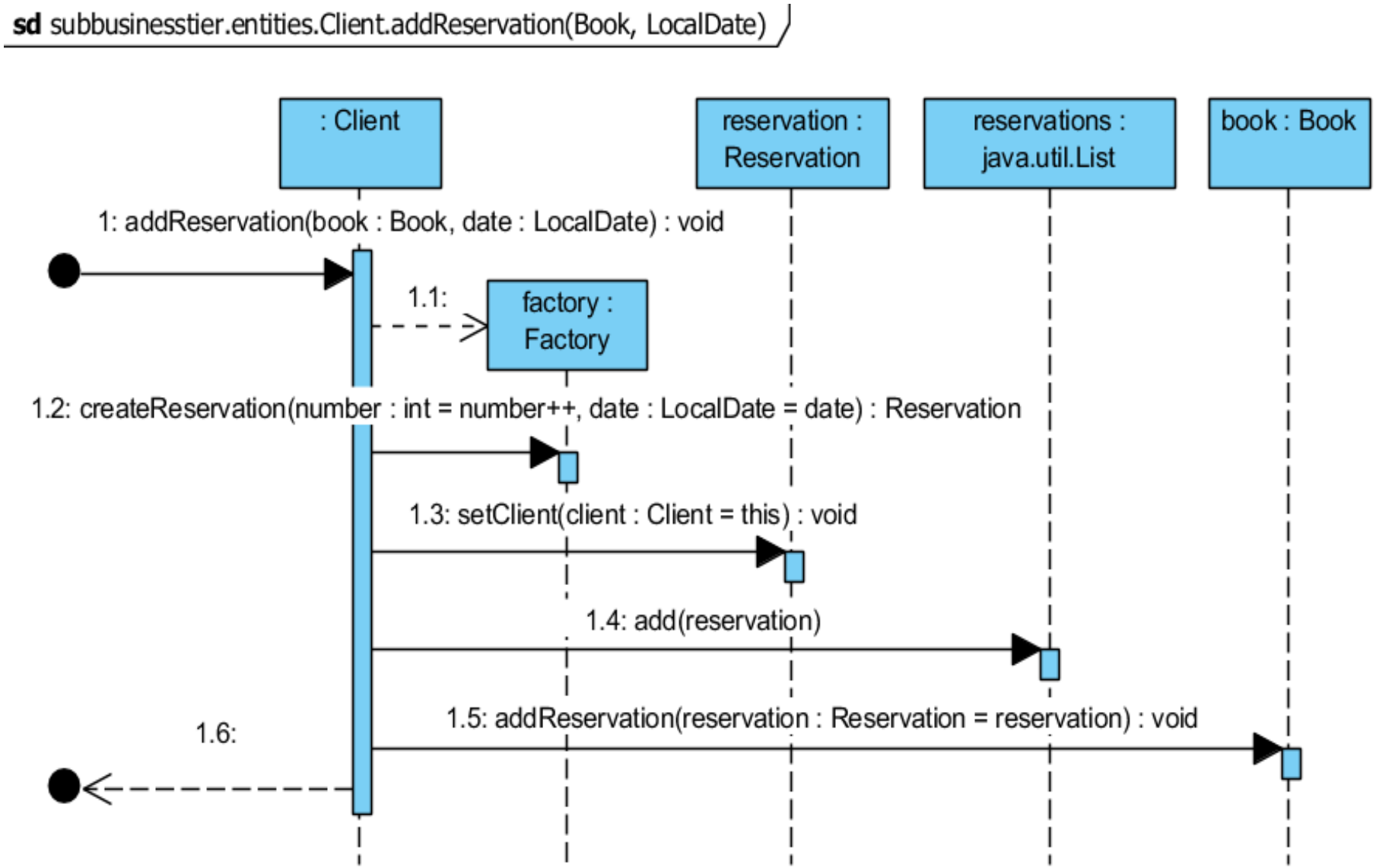
public boolean isFree(LocalDate date)



```
                                    //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

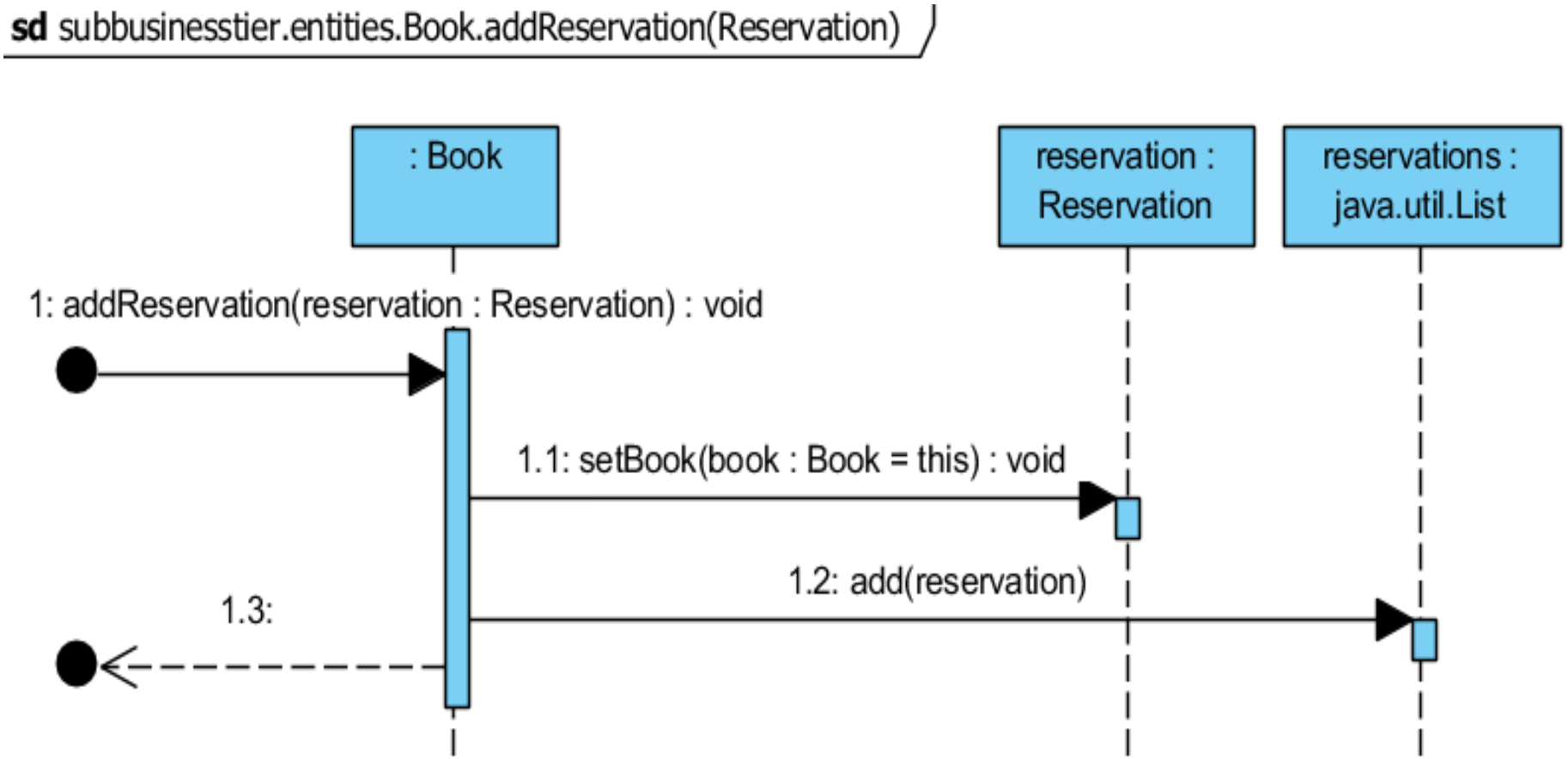
public void addReservation(Book book, LocalDate date)



```
                                    //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);
   }
```

Modelowanie zachowania obiektów za pomocą diagramów sekwencji i aktywności. Diagramy stanów

1. Wprowadzenie
2. Modelowanie aktywności za pomocą diagramów sekwencji i aktywności - porównanie
3. Diagramy stanów UML

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

Diagramy stanów UML 2 – część piąta

Na podstawie

UML 2.0 Tutorial

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

Diagramy stanów

1. **Diagramy stanów UML**

   https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

2. **Przykład diagramów stanów UML – modelowanie wpływu przypadków użycia na stany obiektu**

Diagramy stanów

1. Diagramy stanów UML

https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html

Dwa rodzaje diagramów UML 2

Diagramy UML modelowania strukturalnego

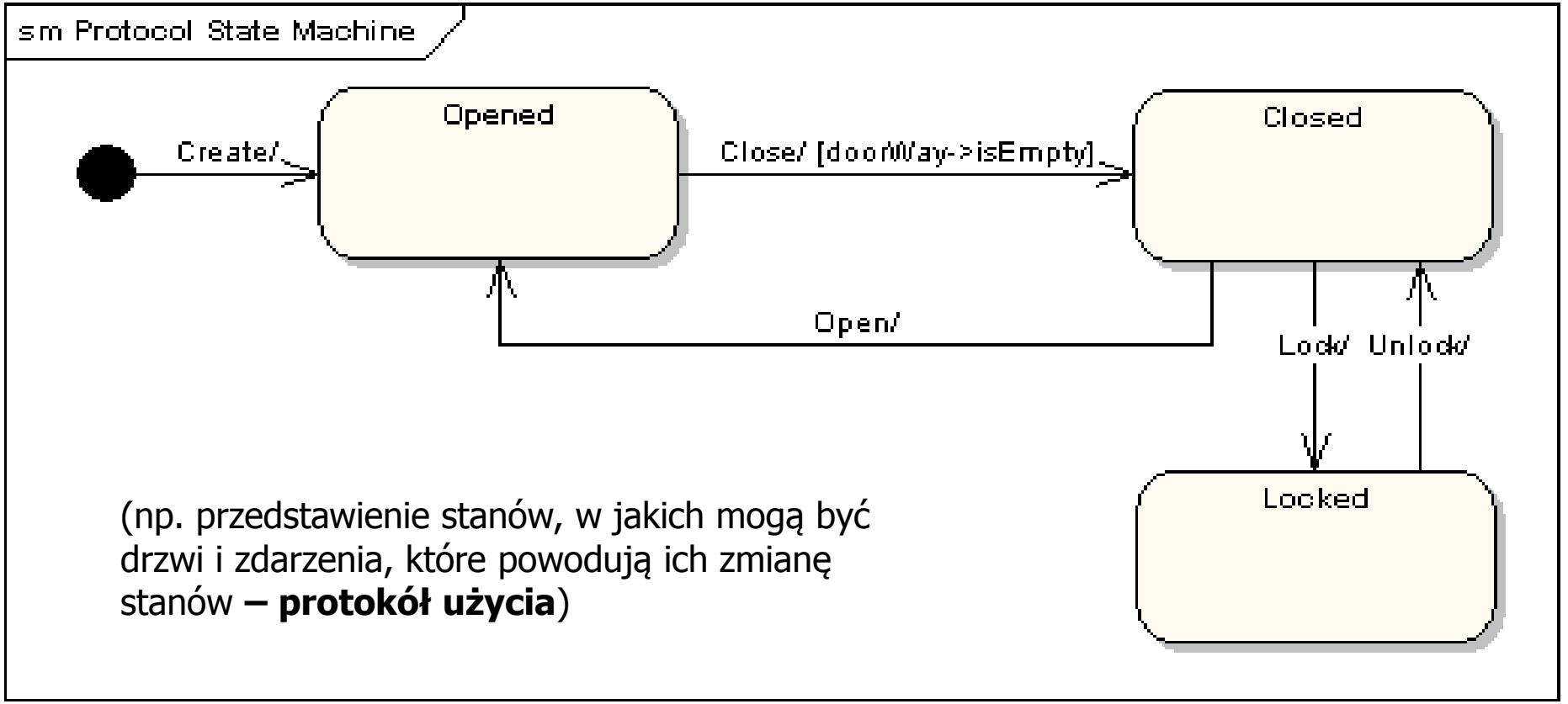
- **Diagramy pakietów**
- ***Diagramy klas***
- **Diagramy obiektów**
- **Diagramy mieszane**
- **Diagramy komponentów**
- **Diagramy wdrożenia**

Diagramy UML modelowania zachowania

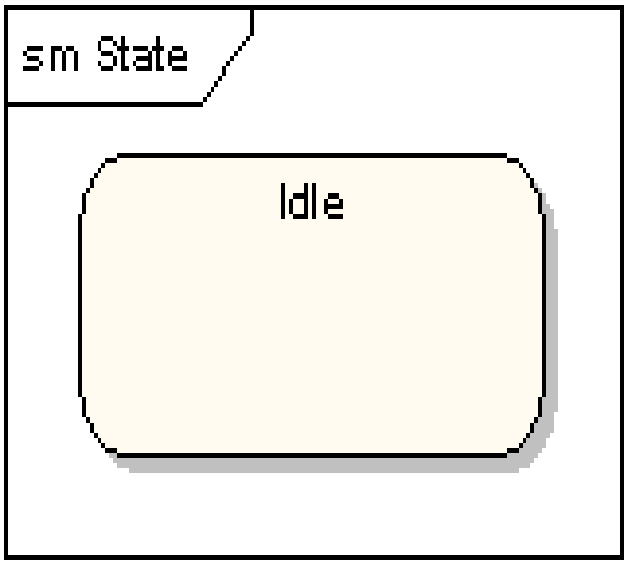
- ***Diagramy przypadków użycia***
- ***Diagramy aktywności***
- ***Diagramy stanów***
- **Diagramy komunikacji**
- ***Diagramy sekwencji***
- **Diagramy czasu**
- **Diagramy interakcji**

Diagram stanu opisuje zmiany stanu obiektu, podsystemu lub systemu pod wpływem działania operacji - iest szczególnie przydatny, gdy zachowanie obiektu jest złożone. Przedstawia on **maszynę stanów** jako przepływ sterowania między stanami.

Diagram stanów jest grafem złożonym z wierzchołków i krawędzi: wierzchołkami są **stany** (prostokąty o zaokrąglonych rogach), krawędziami są **przejścia** (strzałki).

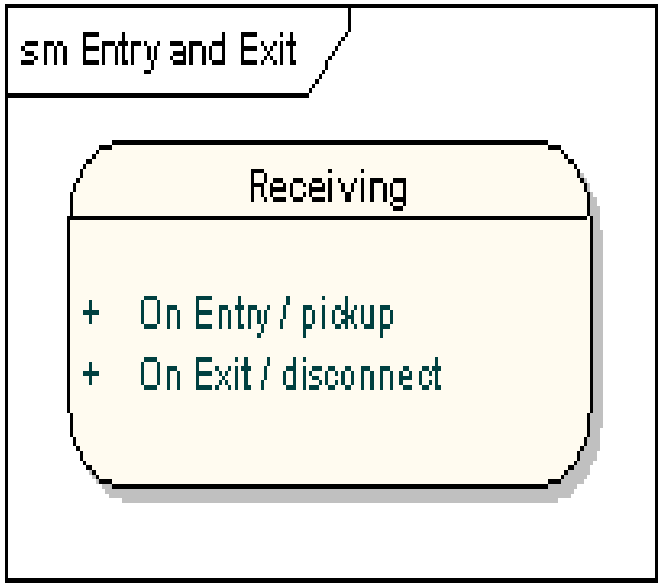


(np. przedstawienie stanów, w jakich mogą być drzwi i zdarzenia, które powodują ich zmianę stanów **– protokół użycia**)



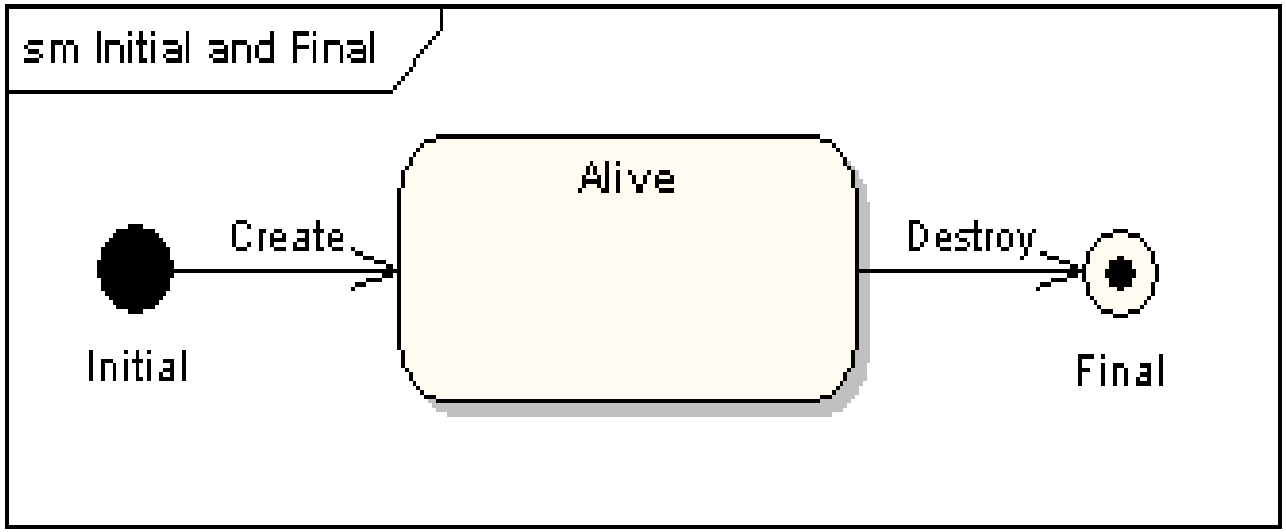
Stan jest okolicznością lub sytuacją, w jakiej znajduje się obiekt

- jest rezultatem poprzedniej aktywności
- spełnia jakiś warunek
- jest określony przez wartości własnych atrybutów i powiązań do innych zadań
- wykonuje pewne czynności
- czeka na jakieś zdarzenie



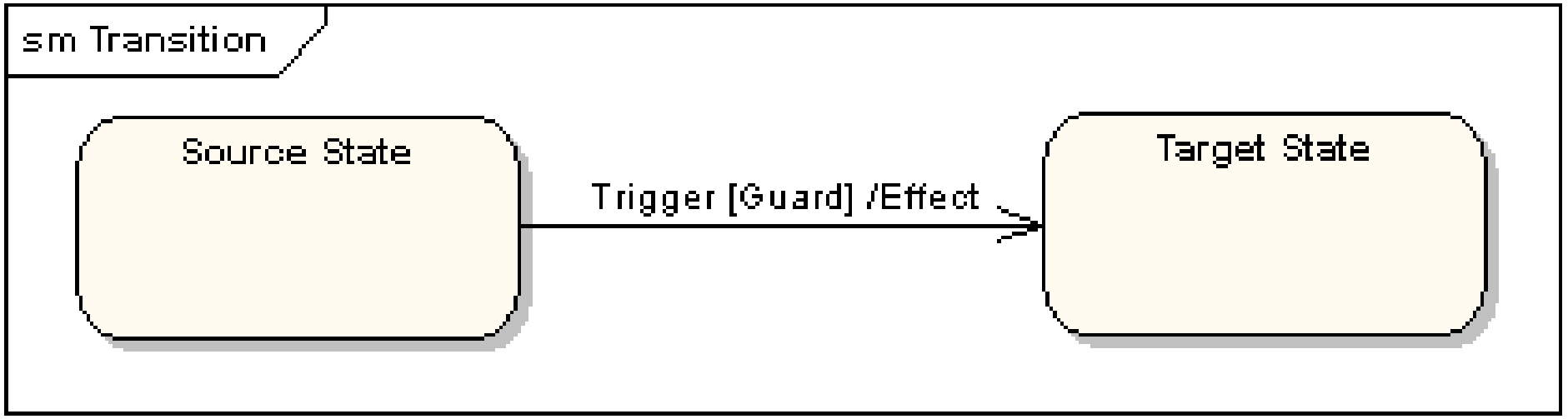
- **Nazwa** -unikatowy ciąg znaków, brak nazwy dla stanu anonimowego

- **Akcje wejściowe** (**entry**) i **wyjściowe** (**exit**) - akcje wykonywane odpowiednio przy wejściu do stanu i przy wyjściu)



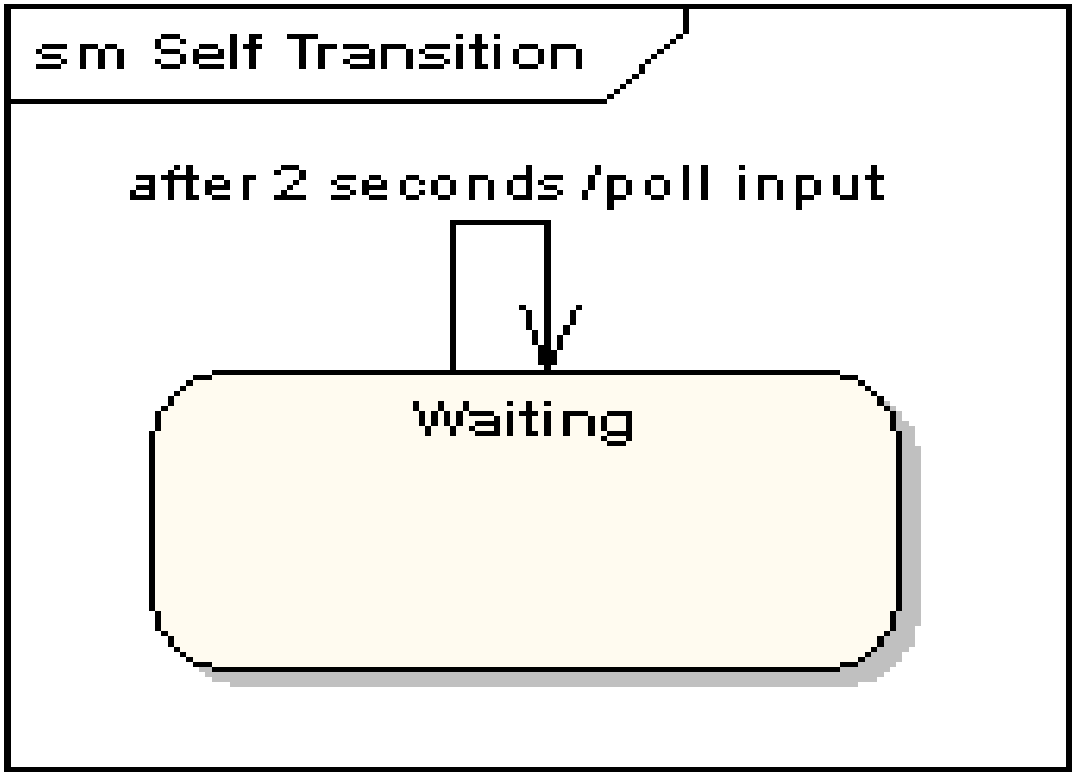
Stan początkowy (**Initial**) – może być tylko jeden stan początkowy

Stan końcowy (**Final**) – może być kilka stanów końcowych



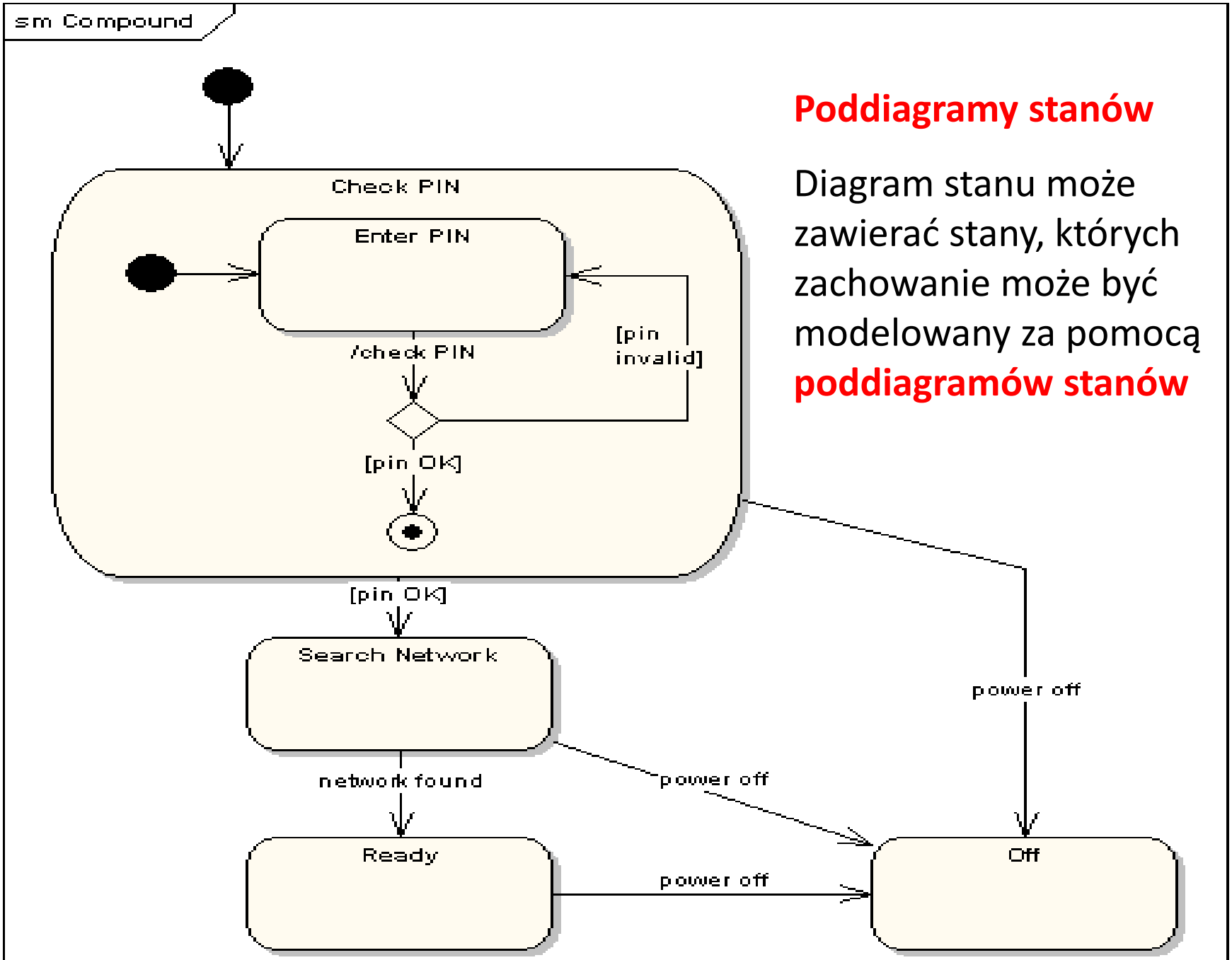
Przejście (**Transition**) jest związkiem między dwoma stanami, który wskazuje, że np. obiekt znajdujący się w pierwszym stanie wykona pewne **akcje (Effect)** i przejdzie do drugiego stanu, ilekroć zaistnieje określone **zdarzenie** (**Trigger**) i będą spełnione określone **warunki (Guard).**

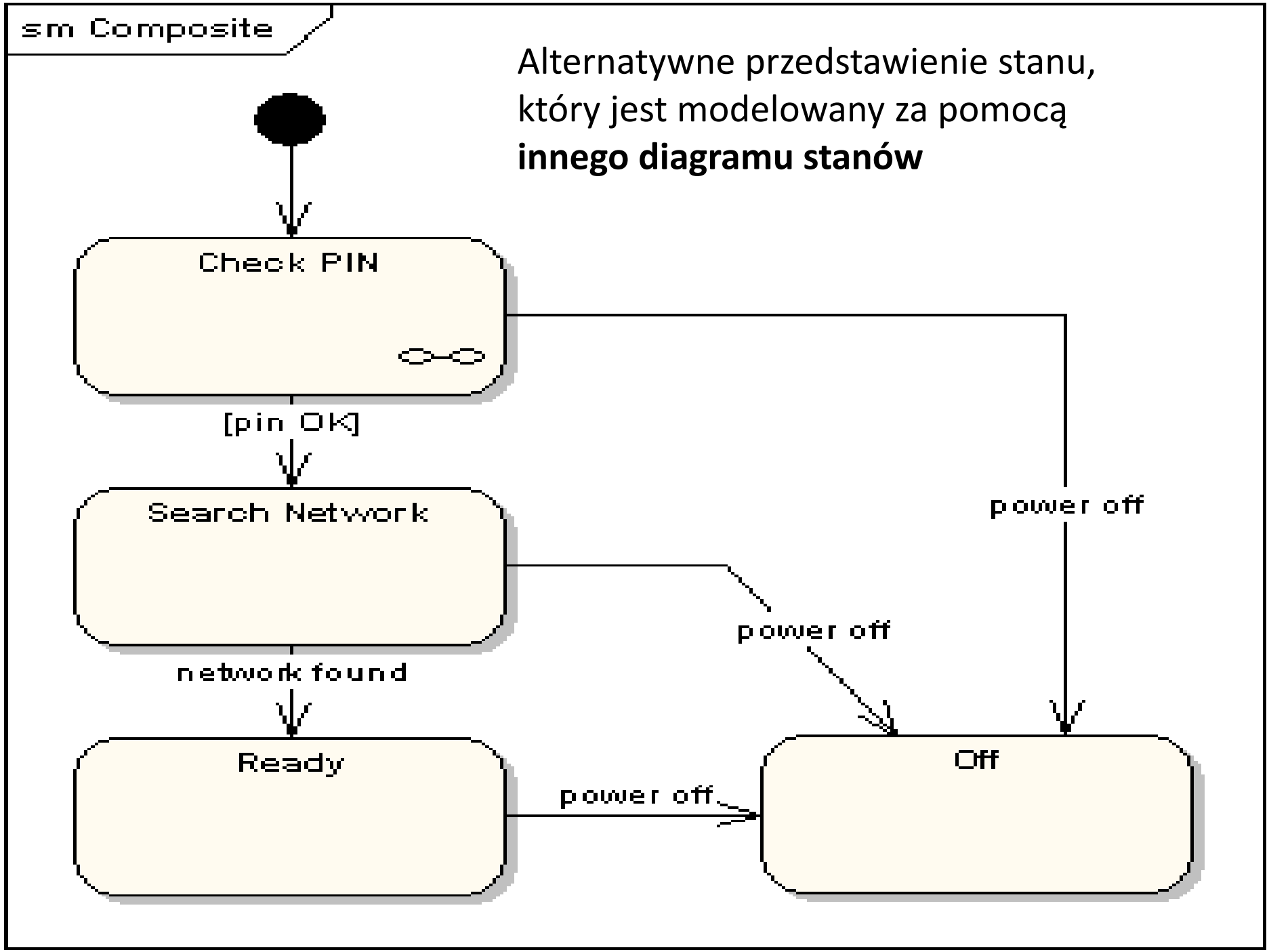
Przejście własne jest związkiem między tym samym stanem, który wskazuje, że np. obiekt znajdujący się w pewnym stanie wykona pewne **akcje** i powróci do tego samego stanu, ilekroć zaistnieje określone **zdarzenie i będą spełnione określone warunki**.

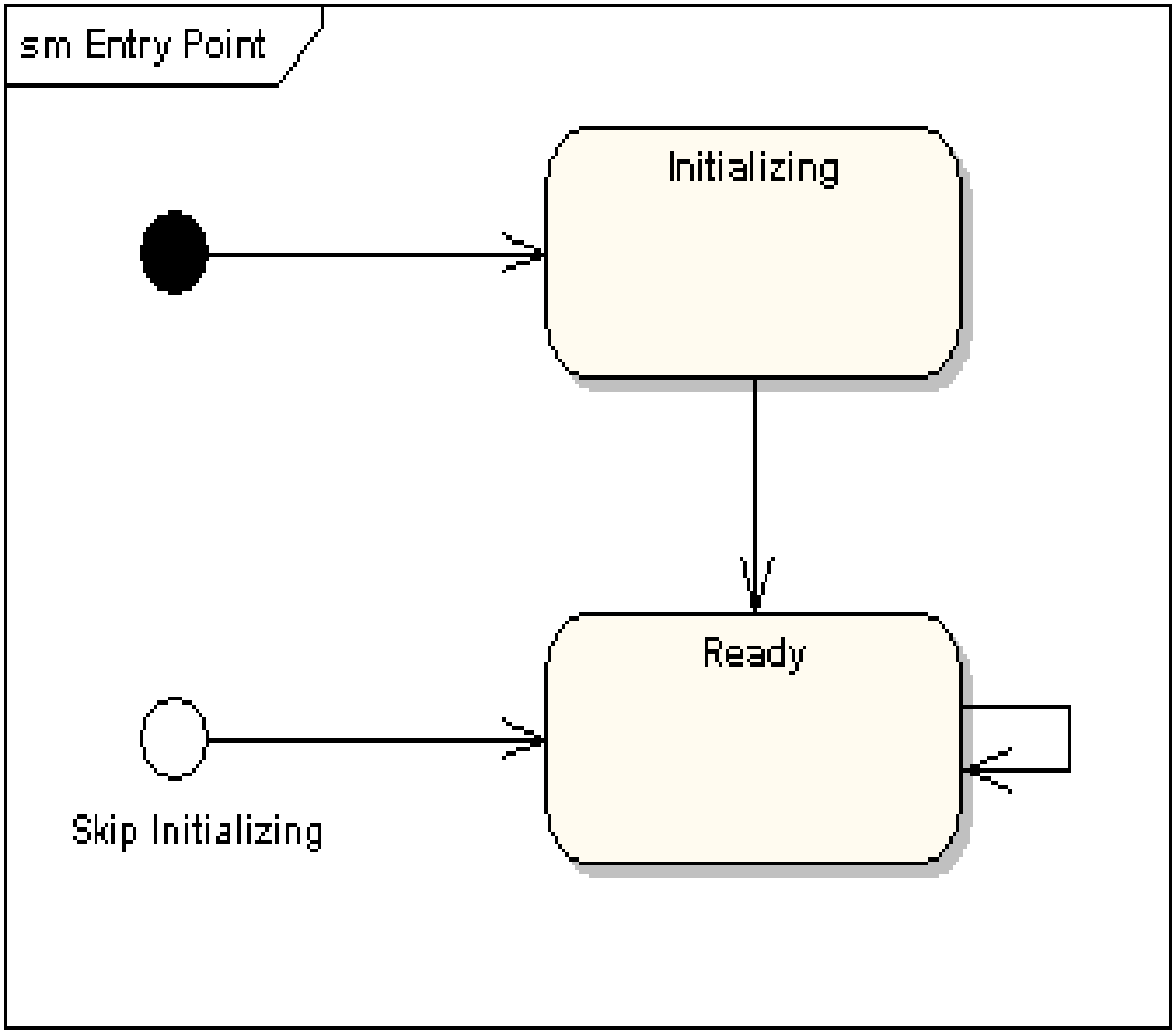


Poddiagramy stanów

Diagram stanu może zawierać stany, których zachowanie może być modelowany za pomocą **poddiagramów stanów**




sm Composite
Alternatywne przedstawienie stanu, który jest modelowany za pomocą **innego diagramu stanów**
Check PIN
[pin OK]
Search Network
network found
Ready
power off
power off
power off
Off



Stany początkowe w poddiagramach stanów

Wskazanie różnych stanów początkowych w **poddiagramie stanów**:

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

Punkty startowe w diagramach nadrzędnych

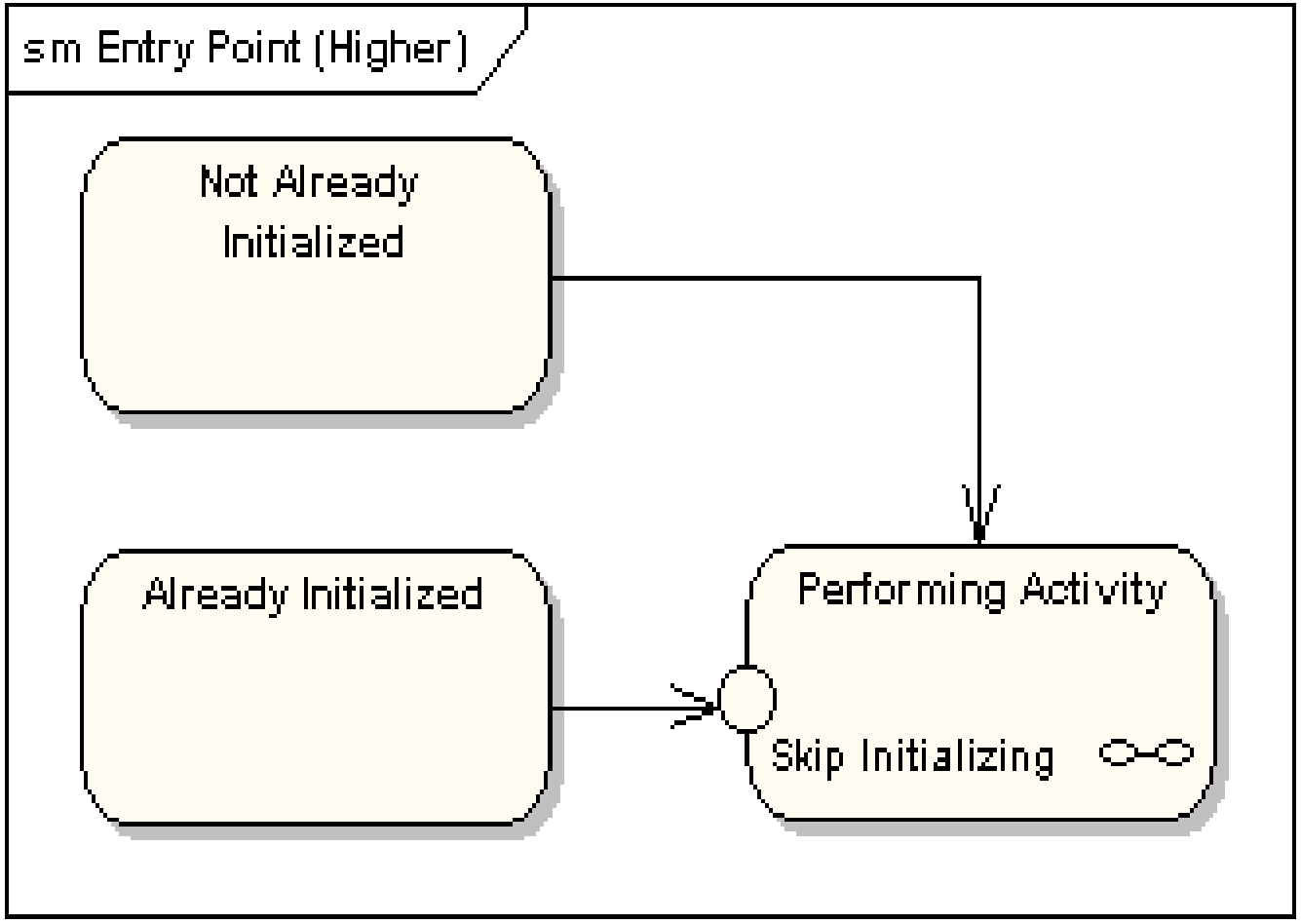
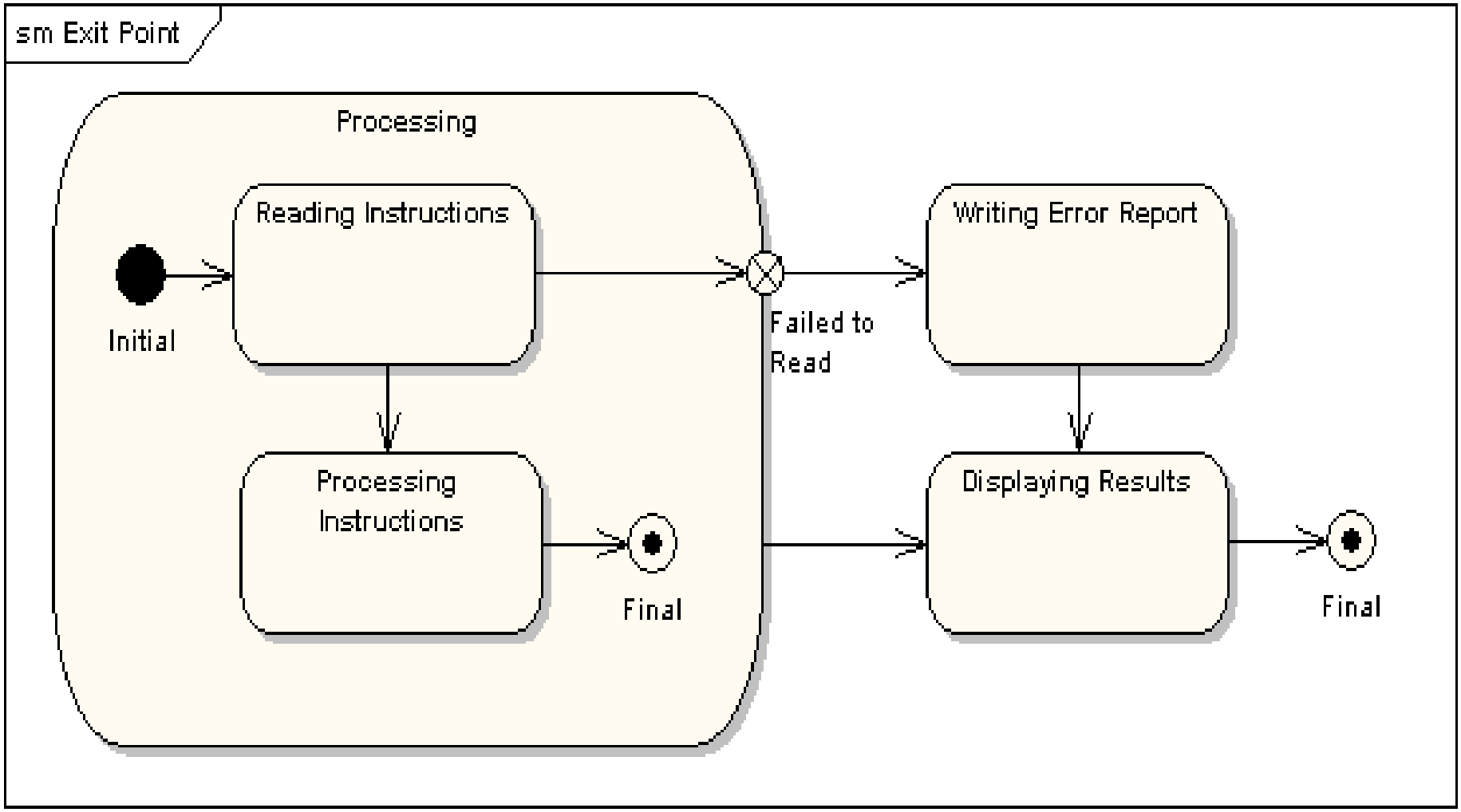


Diagram stanów zawierający różne punkty startowe dla poddiagramów stanów (reprezentowanych przez inne diagramy):

- rozpoczęcie stanu z inicjalizacją (normalne)
- bez inicjalizacji (wyjątkowe)

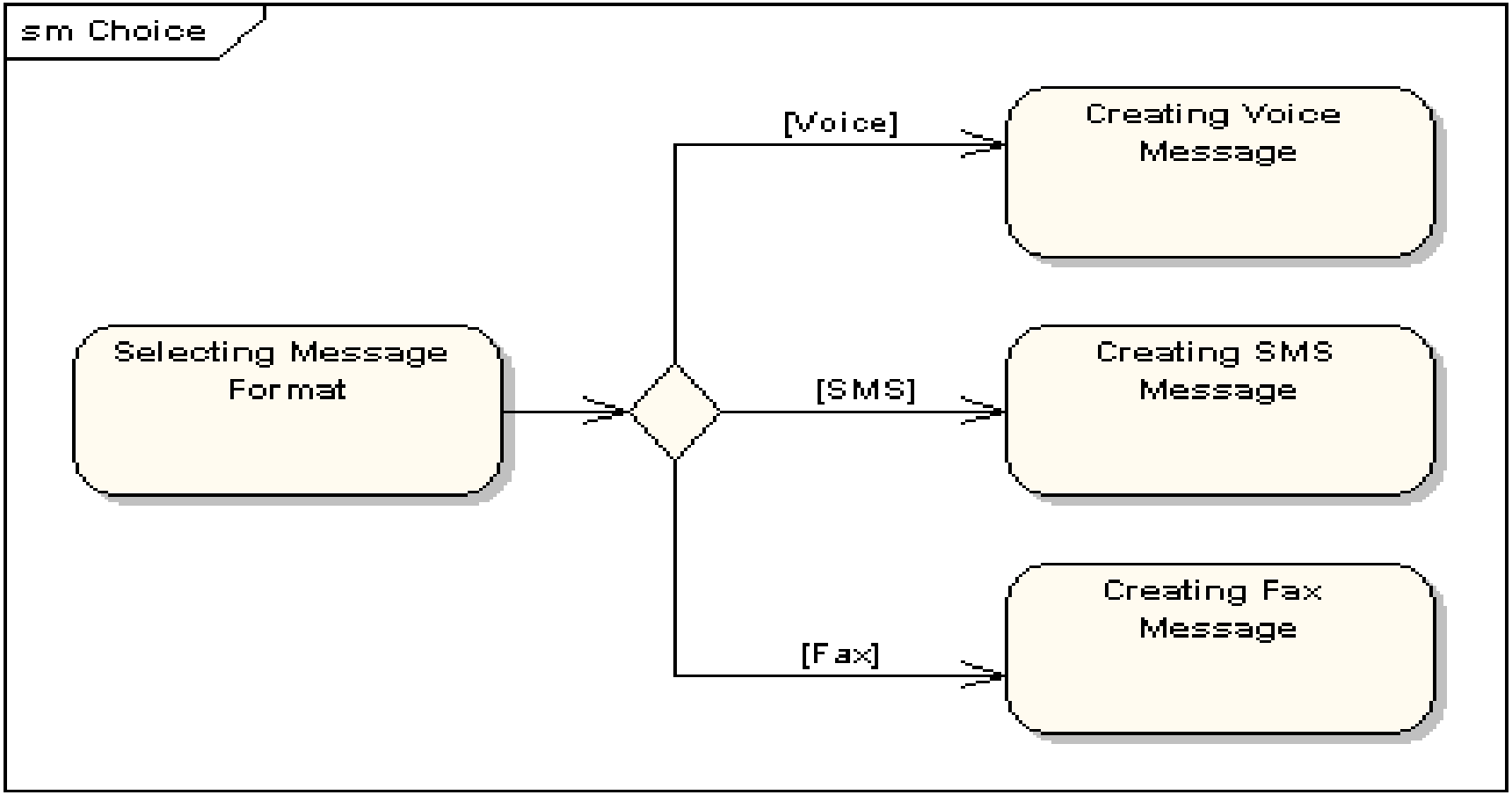
Punkt wyjścia – modelowanie osiągnięcia alternatywnych stanów końcowych (**Final**) przez obiekt



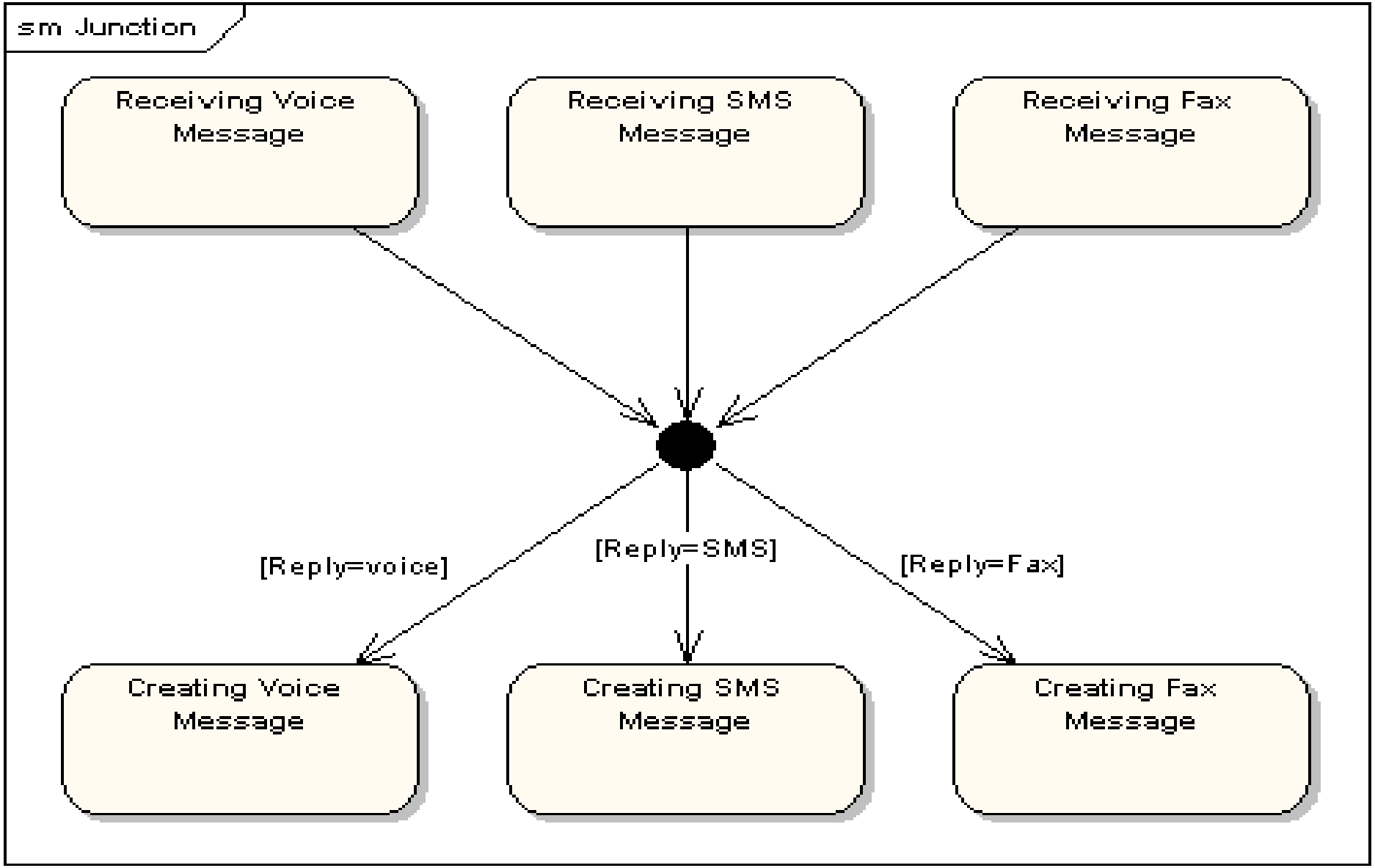
Pseudo stan wyboru:

• jedno przejście ze stanu wejściowego do **pseudo stanu wyboru (romb) i kilka przejść na wyjściu tego pseudo stanu**

•**w wyniku zdarzenia następuje przejście ze stanu wejściowego** (np. Selecting Message Format) **i na podstawie spełnionego warunku wybór przejścia do jednego ze stanów wyjściowych** (np. wybór przejścia na podstawie wybranego formatu wiadomości w stanie wejściowym); dynamiczny charakter wyboru przejścia

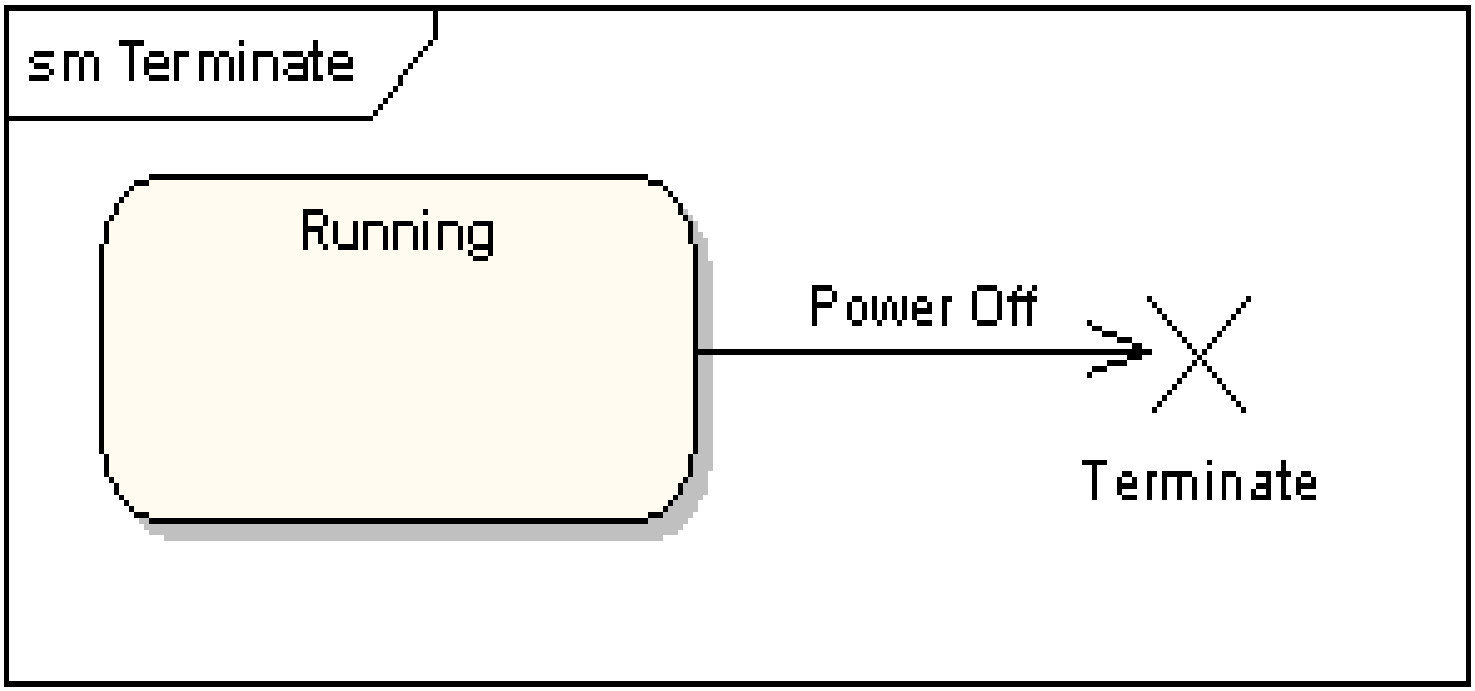


Pseudo stan typu połączenie – w pseudo stanie typu połączenie możliwość wyboru przejść do stanów wyjściowych po zdarzeniach zachodzących na przejściach ze stanów wejściowych

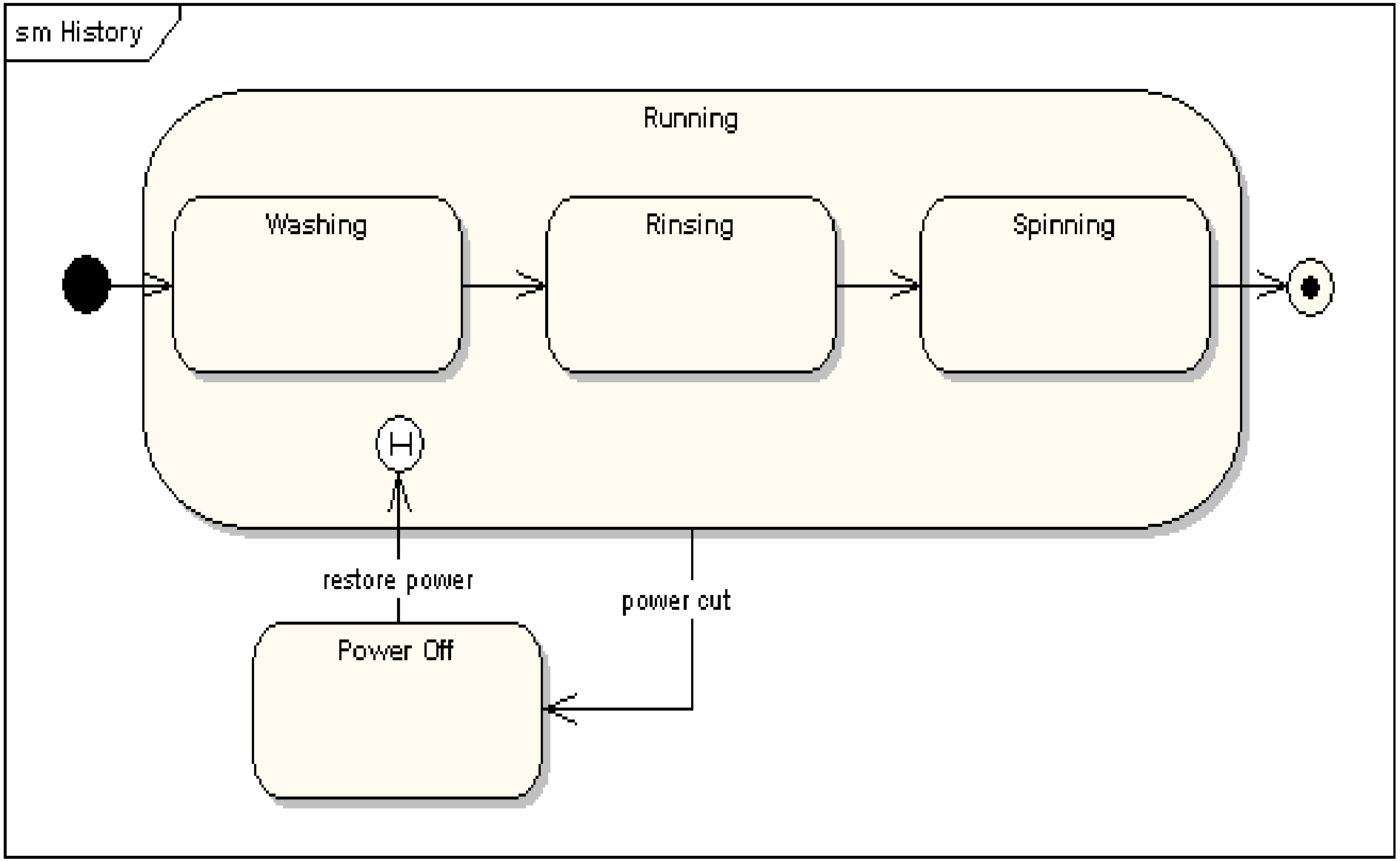


Pseudo stan typu zakończenie

oznacza zakończenie linii życia maszyny stanowej



Stany historyczne – przedstawiają stany wcześniejsze (historyczne) przed przerwaniem działania maszyny stanowej (np. w chwili załączenia zasilania maszyna stanowa zmywarki pamięta stan, w którym ma wznowić działanie)



Równolegle podstany

Stan może być podzielony między równoległe podstany wykonywane jednocześnie. (np. sterowanie przednimi (front) i tylnymi (rear) hamulcami odbywa się równolegle i musi być zsynchronizowane – wyrażone za pomocą symbolu rozdzielenia na pseudo stany oraz symbolu połączenia pseudo stanów. Równoległe podstany są używane do modelowania synchronizacji wątków

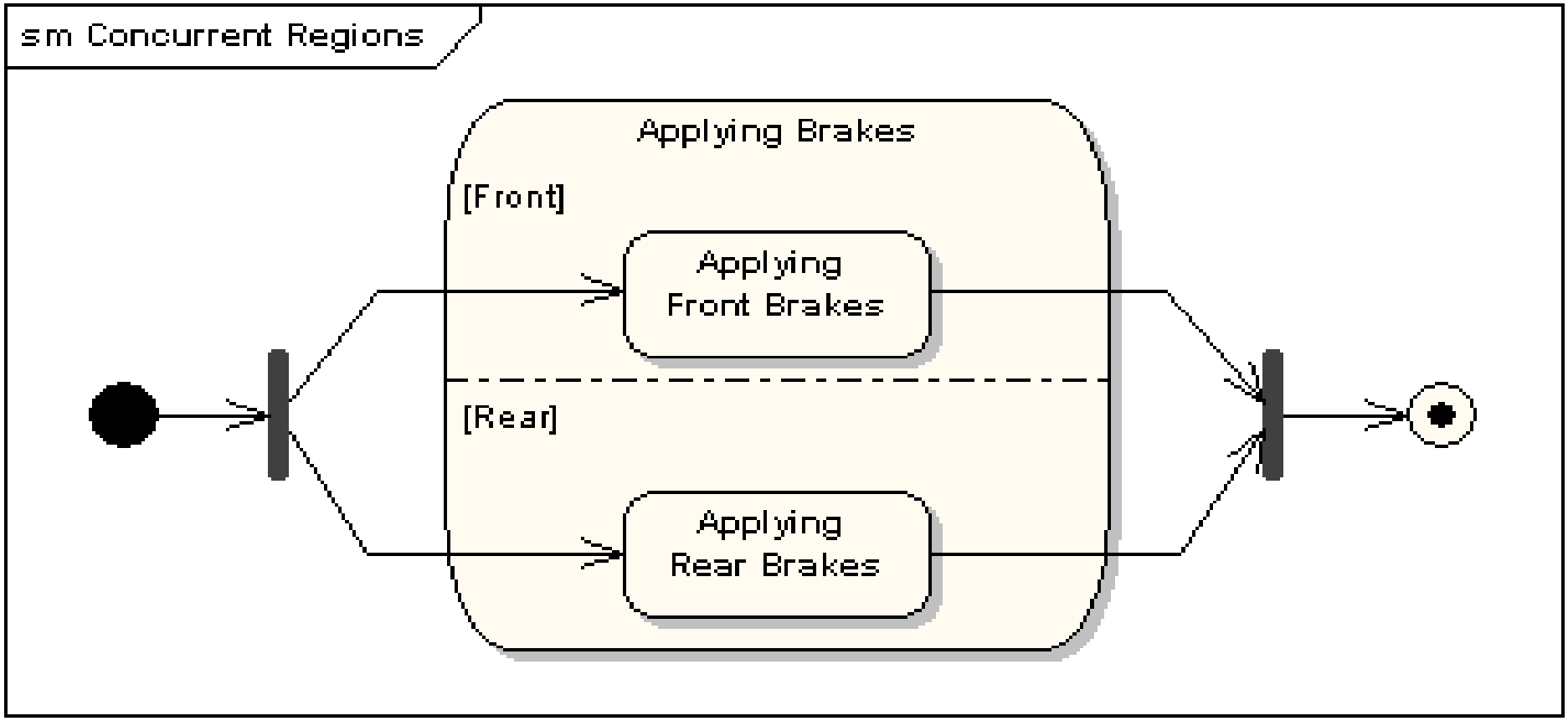


Diagram stanów

1. **Diagramy stanów UML**
   https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html
2. **Przykład diagramu stanów UML – modelowanie wpływu przypadków użycia na stany obiektu**

Diagram stanów klasy *TitleBook*

Zdarzenia: equals, addBook, searchFreeBook, getBook

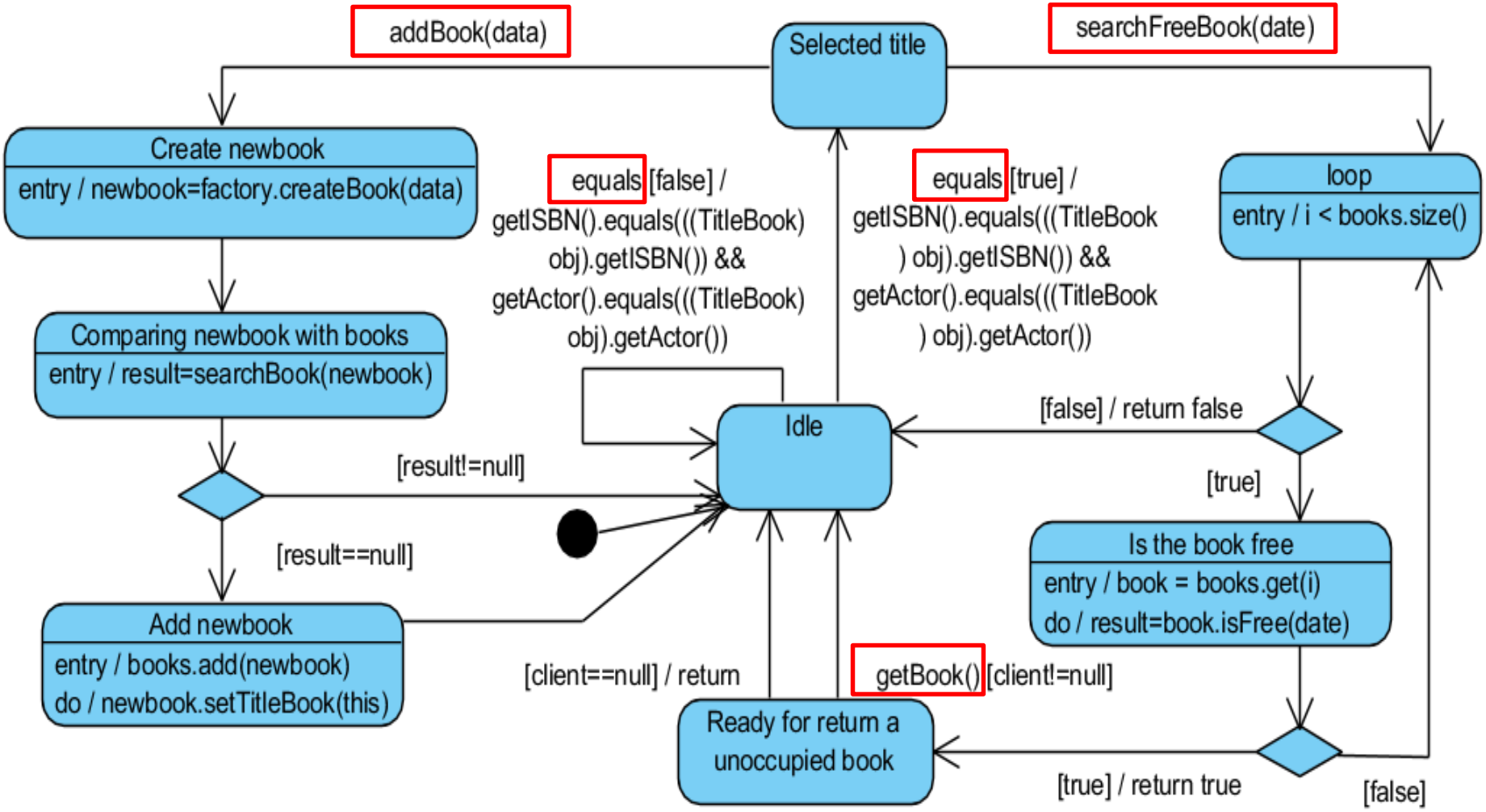


Diagram przypadków użycia (wykład 4 część 1, przykład 3) – wybrany fragment

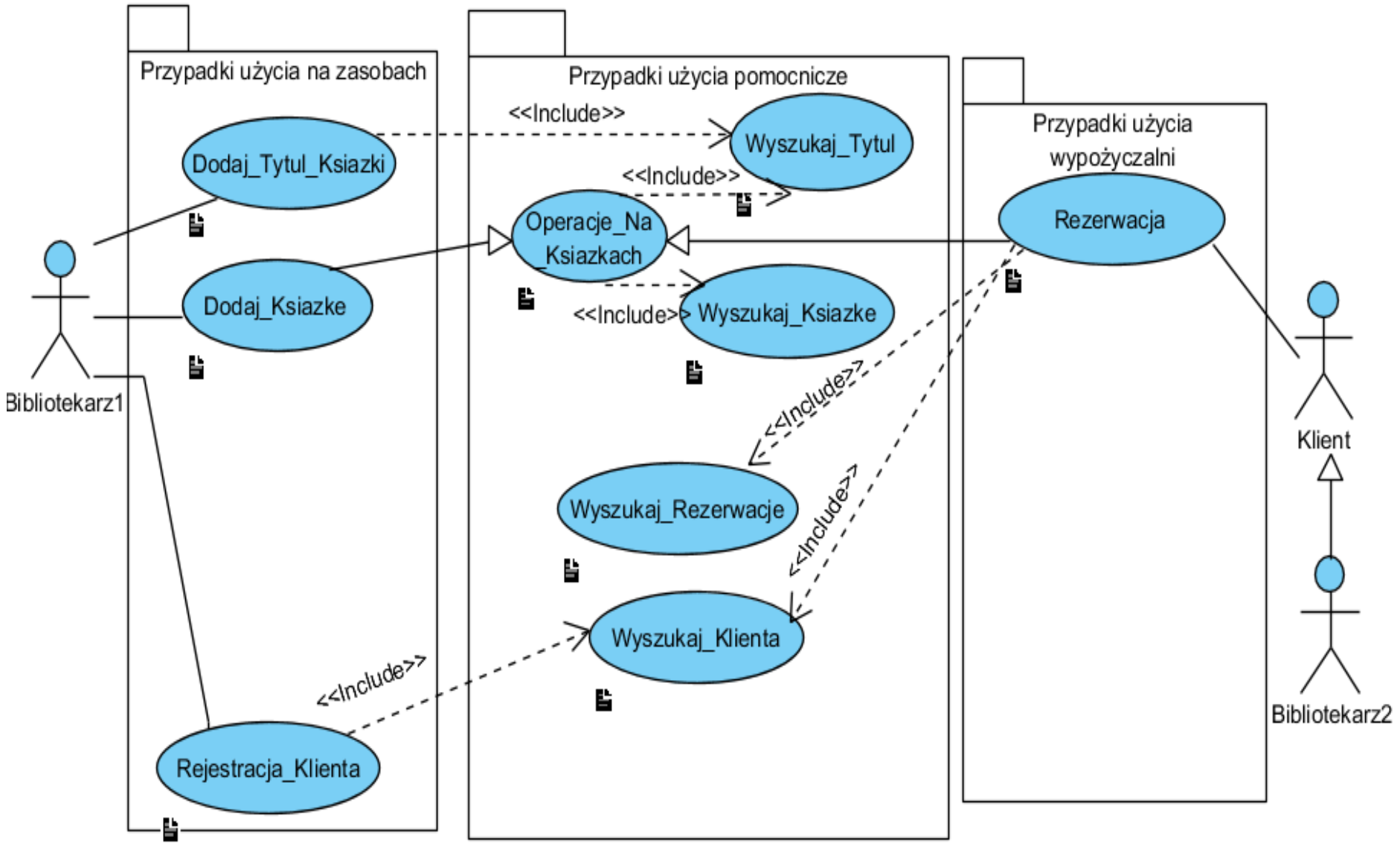
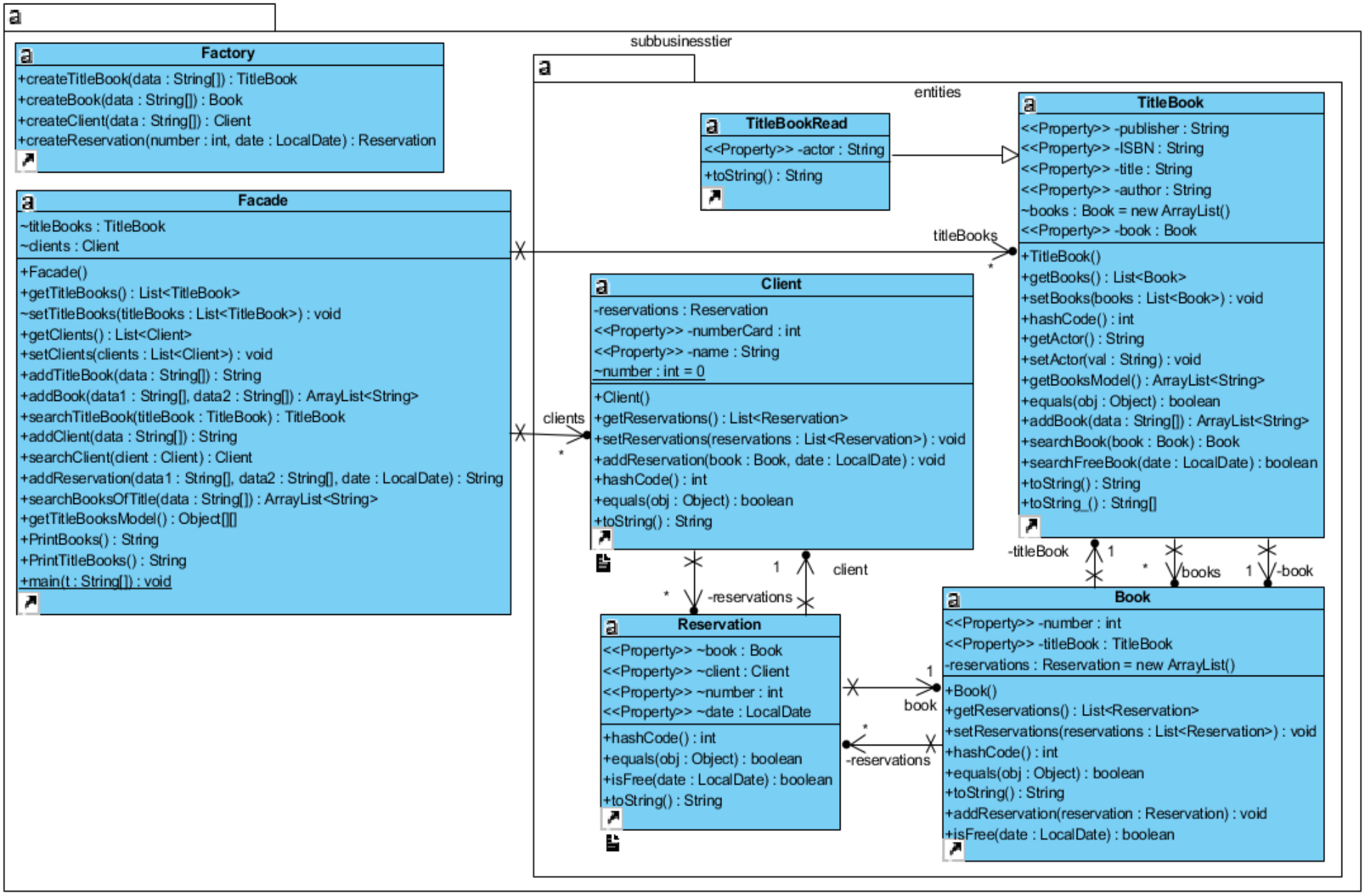


Diagram klas uzyskany w procesie projektowania (przebieg pokazany w dodatku do wykładu 5)



Klasa Facade udostępnia metody logiki biznesowej – generuje bezpośrednio 3 zdarzenia na obiektach z rodziny TitleBook przez wywołanir jego metod: addBook, searchFreeBook, getBook oraz 1 zdarzenie generuje pośrednio: equals

```
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)             {  }
```

Zdarzenia wywołane na obiektach z rodziny TitleBook przez obiekt typu Facade oraz jego atrybut titleBooks

equals, addBook, searchFreeBook, getBook

```
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

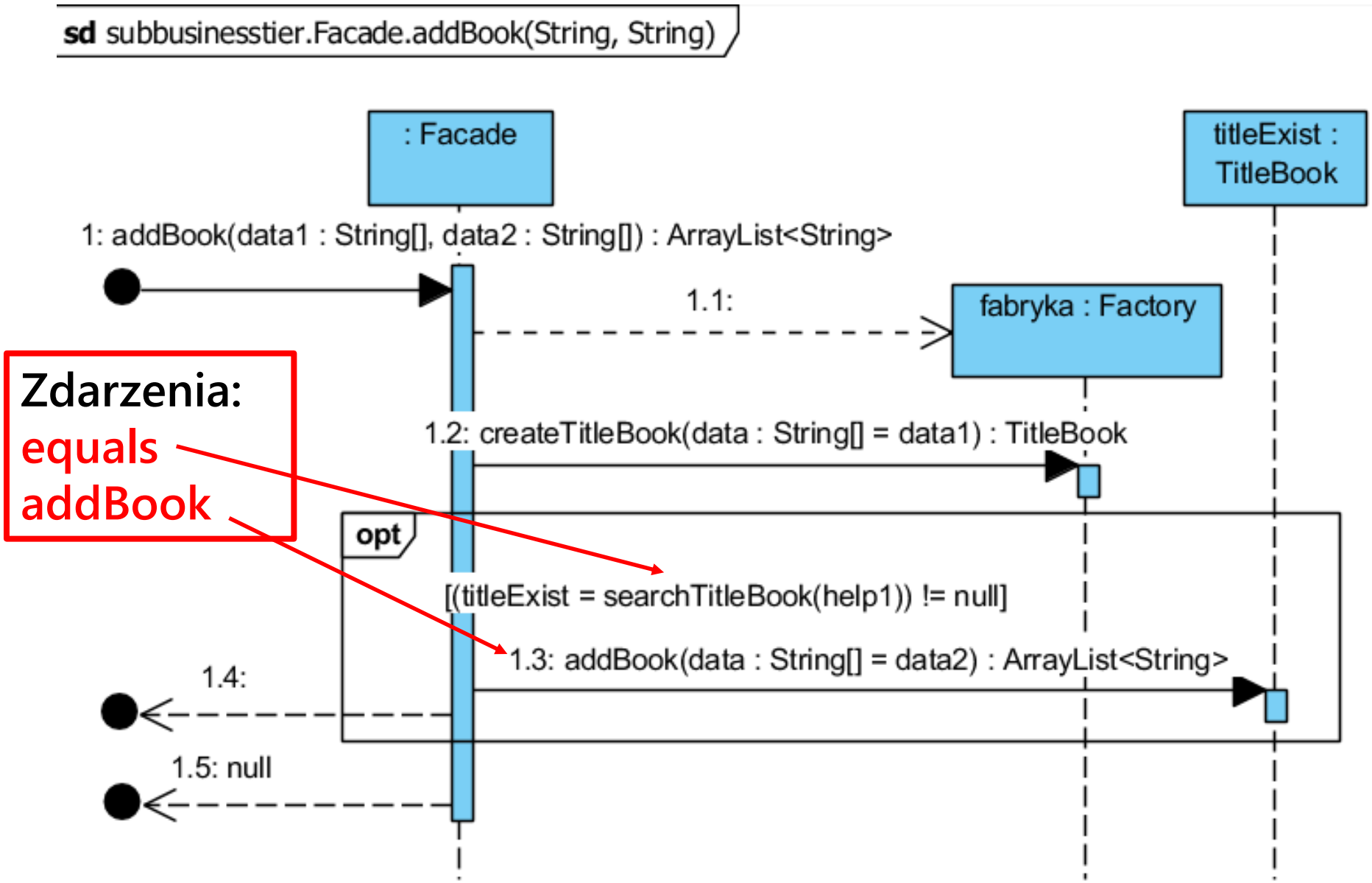
PU Rejestracja_Klienta

PU Dodaj_Tytul_Ksiazki

PU Dodaj_Ksiazke

PU Rezerwacja

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

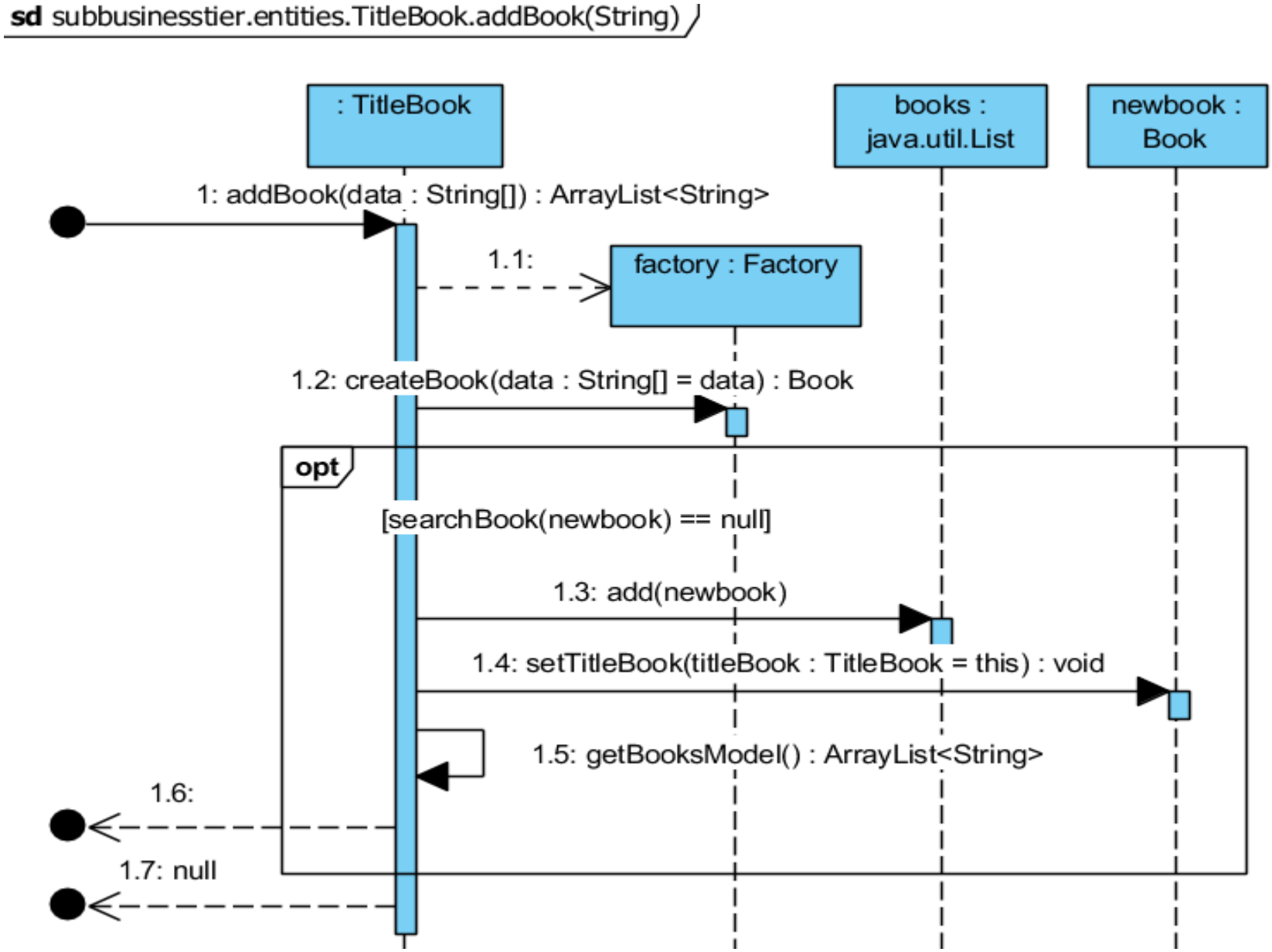


```
//class Facade
  List<TitleBook> titleBooks;
  List<Client> clients;
  public Facade() {
     titleBooks = new ArrayList<>();
     clients = new ArrayList();
  }

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

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

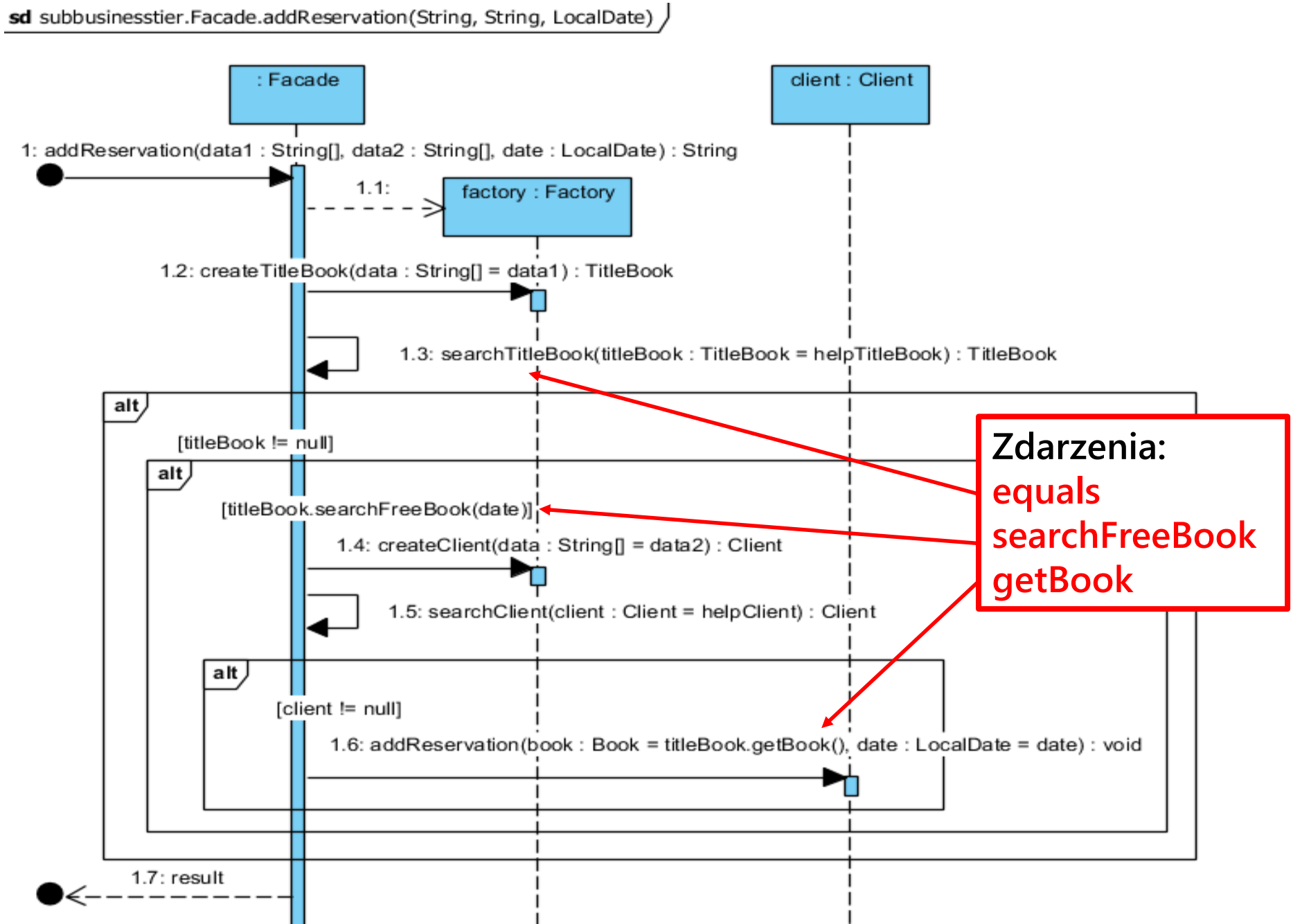
```
public int indexOf(Object o) {
    if (o == null) {
      for (int i = 0; i < size; i++)
        if (elementData[i]==null)
          return i;
    } else {
      for (int i = 0; i < size; i++)
        if (o.equals(elementData[i]))
          return i;  }
    return -1;  }
```

Wiadomości metody addBook wywołanej jako zdarzenie na obiekcie z rodziny TitleBook przez obiekt typu Facade - odwzorowane na akcje na diagramie stanów



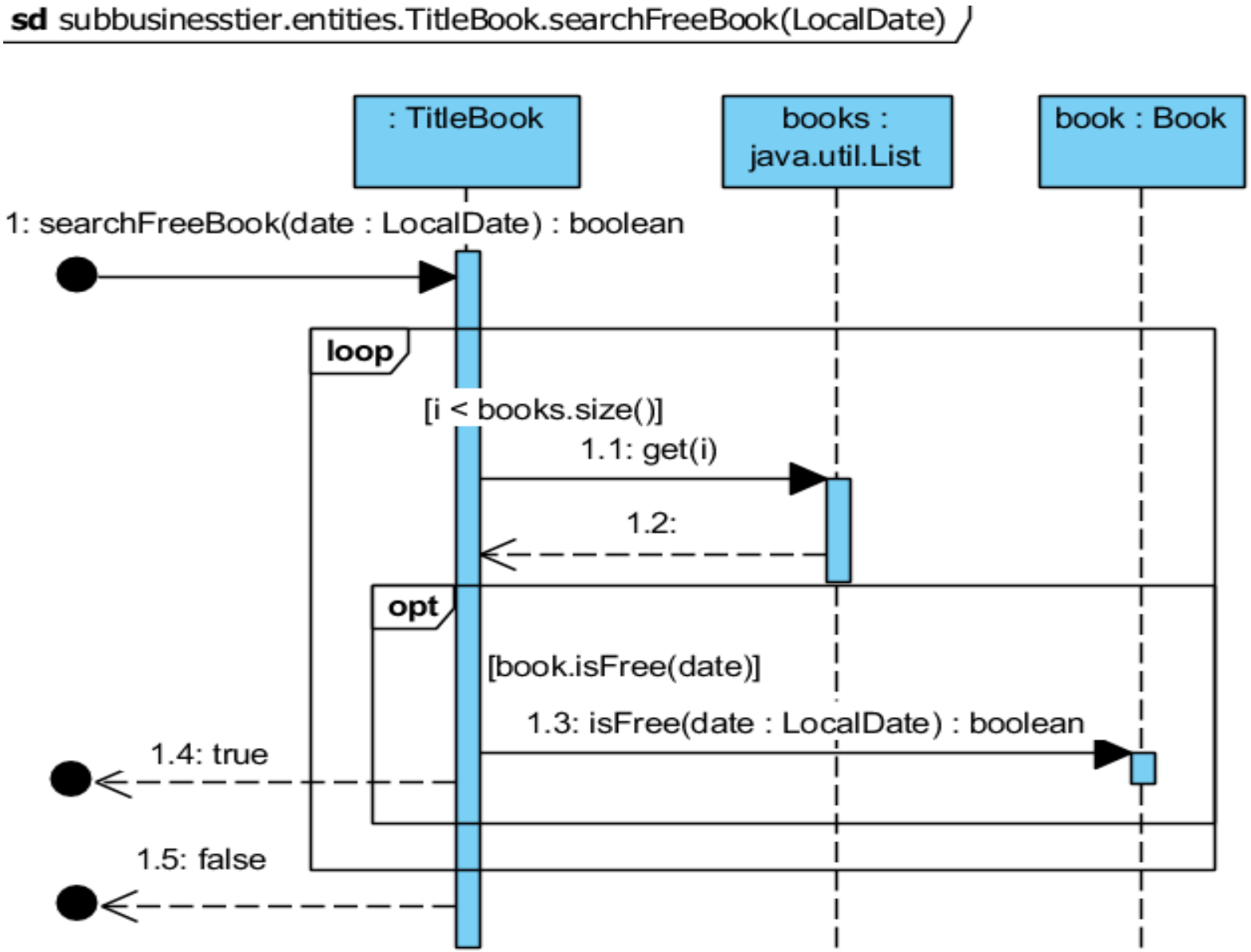
```
//class TitleBook
  List<Book> books;
  public TitleBook() {
     books = new ArrayList();
  }

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;
  }
```

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



```
                                    //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)) {
          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;
 }
```

Wiadomości metody searchFreeBook wywołanej jako zdarzenie na obiekcie z rodziny TitleBook przez obiekt typu Facade – odwzorowane na akcje na diagramie stanów



```
//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;
   }
```