Sequence diagrams

Quizz questions handling concurrency

You are given this sequence diagram. Which of the following traces are possible?

a) $c \longrightarrow a \longrightarrow b$

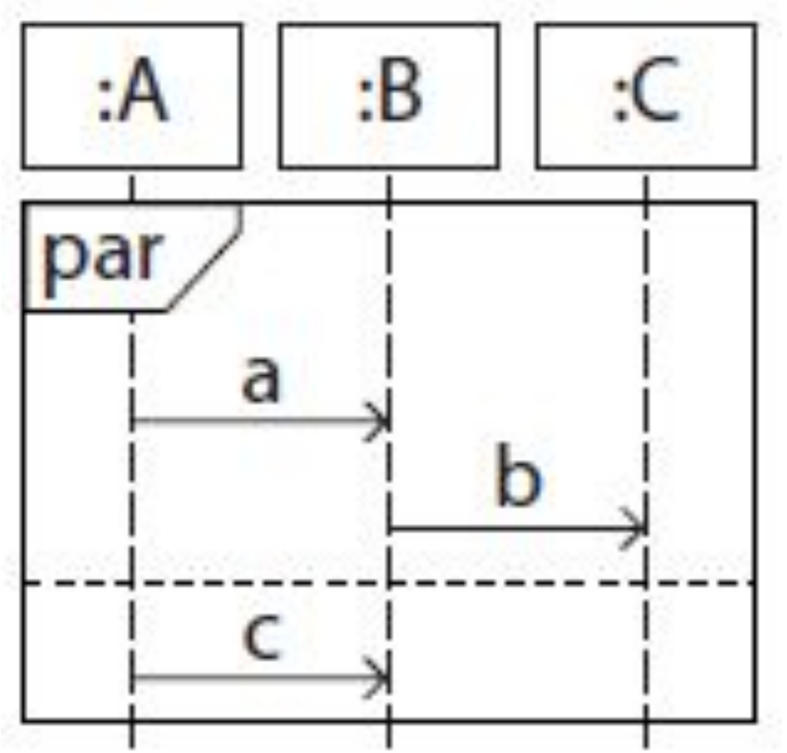
b) $c \longrightarrow b \longrightarrow a$

c) $a \longrightarrow b \longrightarrow c$

d) $b \longrightarrow a \longrightarrow c$

e) $a \longrightarrow c \longrightarrow b$

f) $b \longrightarrow c \longrightarrow a$



You are given this sequence diagram. Which of the following traces are possible?

a) c → a → b

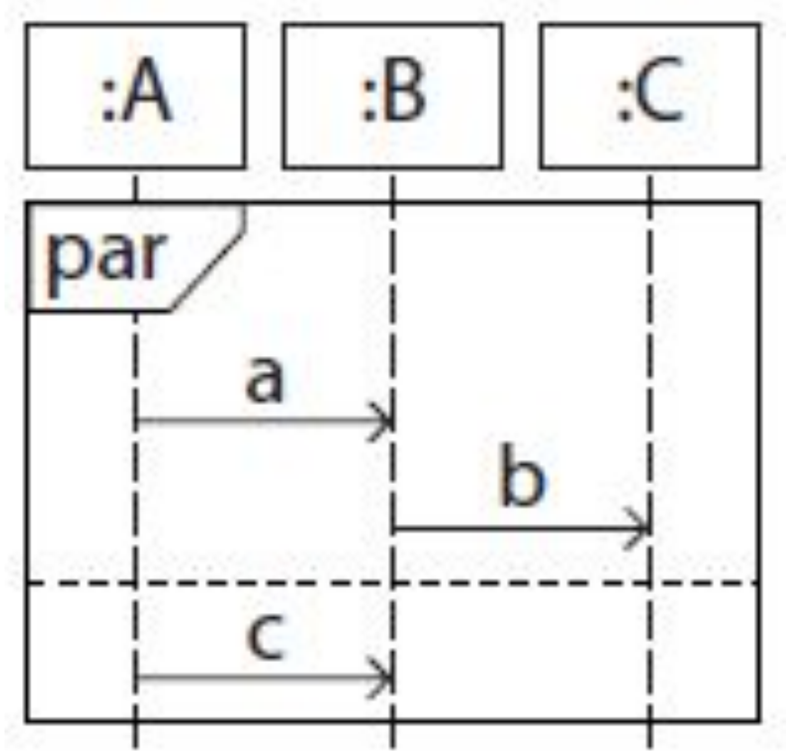
b) c → **b** → **a**

c) a → b → c

d) **b** → **a** → c

e) a → c → b

f) **b** → c → **a**



b can’t be before a

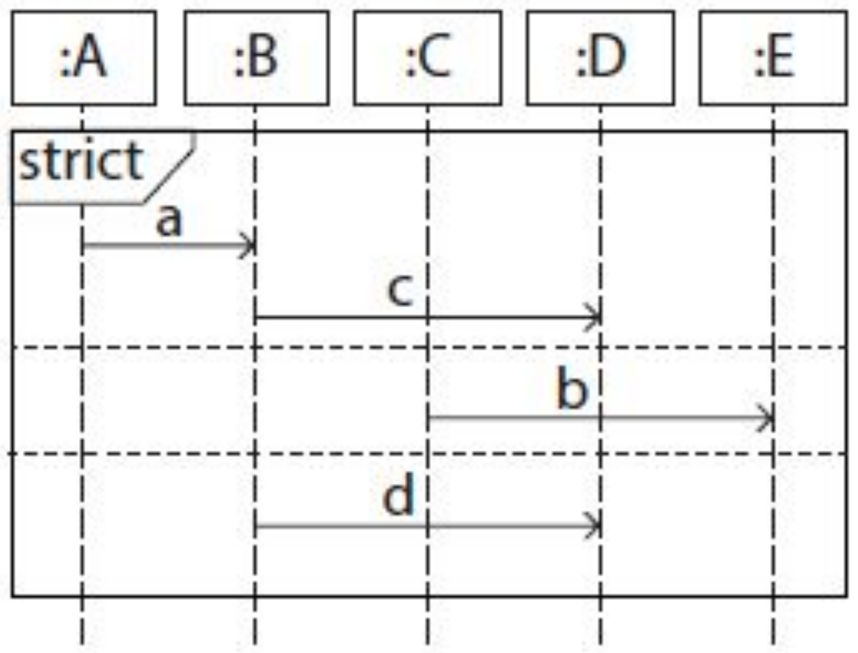
You are given this sequence diagram. Which of the following traces are possible?

a) a ⟶ c ⟶ b ⟶ d

b) a ⟶ b ⟶ d ⟶ c

c) a ⟶ b ⟶ c ⟶ d

d) b ⟶ d ⟶ a ⟶ c



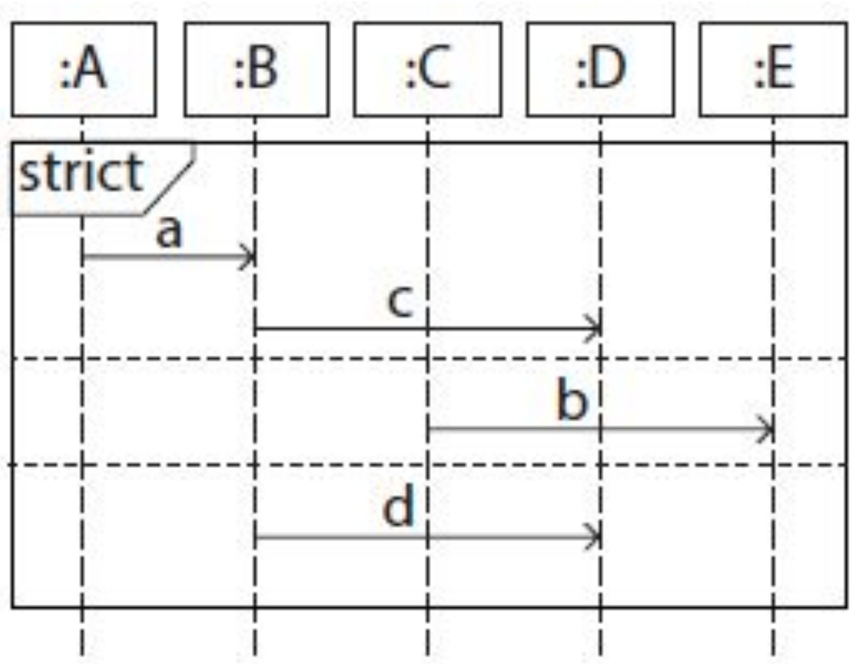
You are given this sequence diagram.
Which of the following traces are possible?

a) a ⟶ c ⟶ b ⟶ d

b) **a** ⟶ b ⟶ d ⟶ **c**

c) **a** ⟶ b ⟶ **c** ⟶ d

d) b ⟶ d ⟶ **a** ⟶ **c**



You are given this sequence diagram. Which of the following traces are possible?

a) $a \rightarrow b \rightarrow c \rightarrow e \rightarrow d$

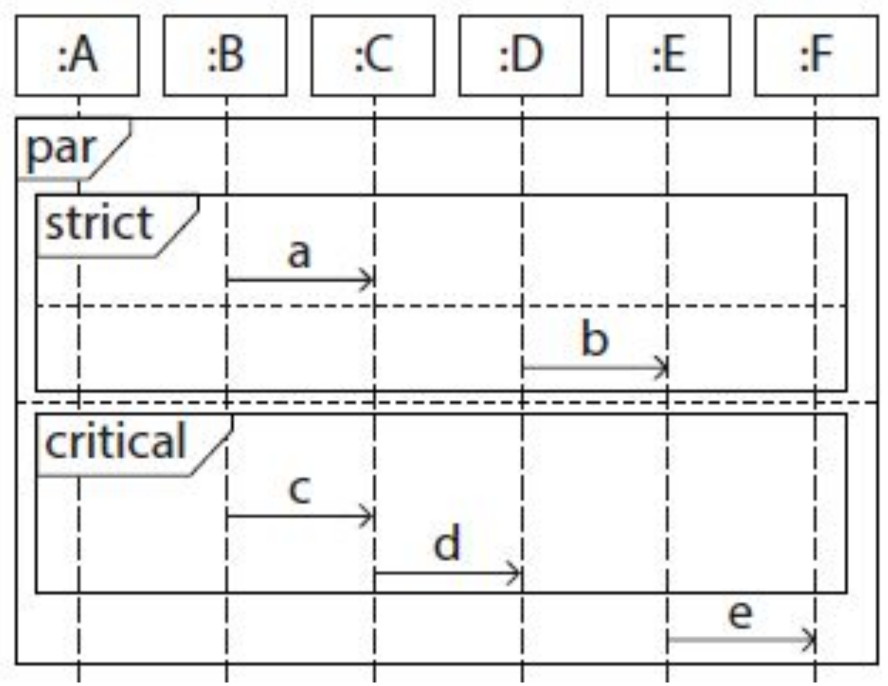
b) $c \rightarrow d \rightarrow a \rightarrow b \rightarrow e$

c) $a \rightarrow c \rightarrow d \rightarrow b \rightarrow e$

d) $e \rightarrow a \rightarrow b \rightarrow c \rightarrow d$

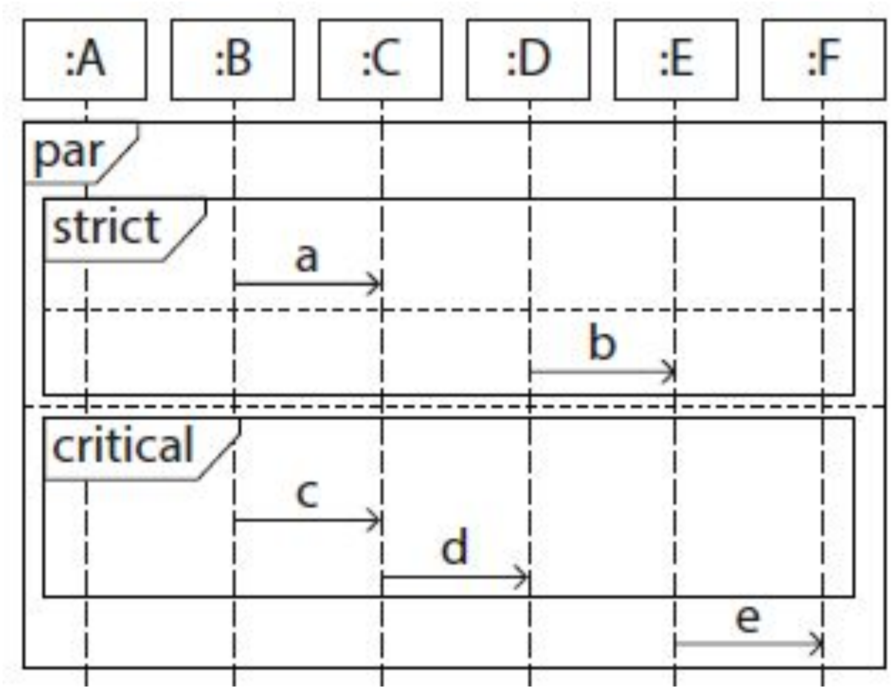
e) $c \rightarrow a \rightarrow e \rightarrow d \rightarrow b$

f) $a \rightarrow b \rightarrow e \rightarrow d \rightarrow c$



You are given this sequence diagram.
Which of the following traces are possible?

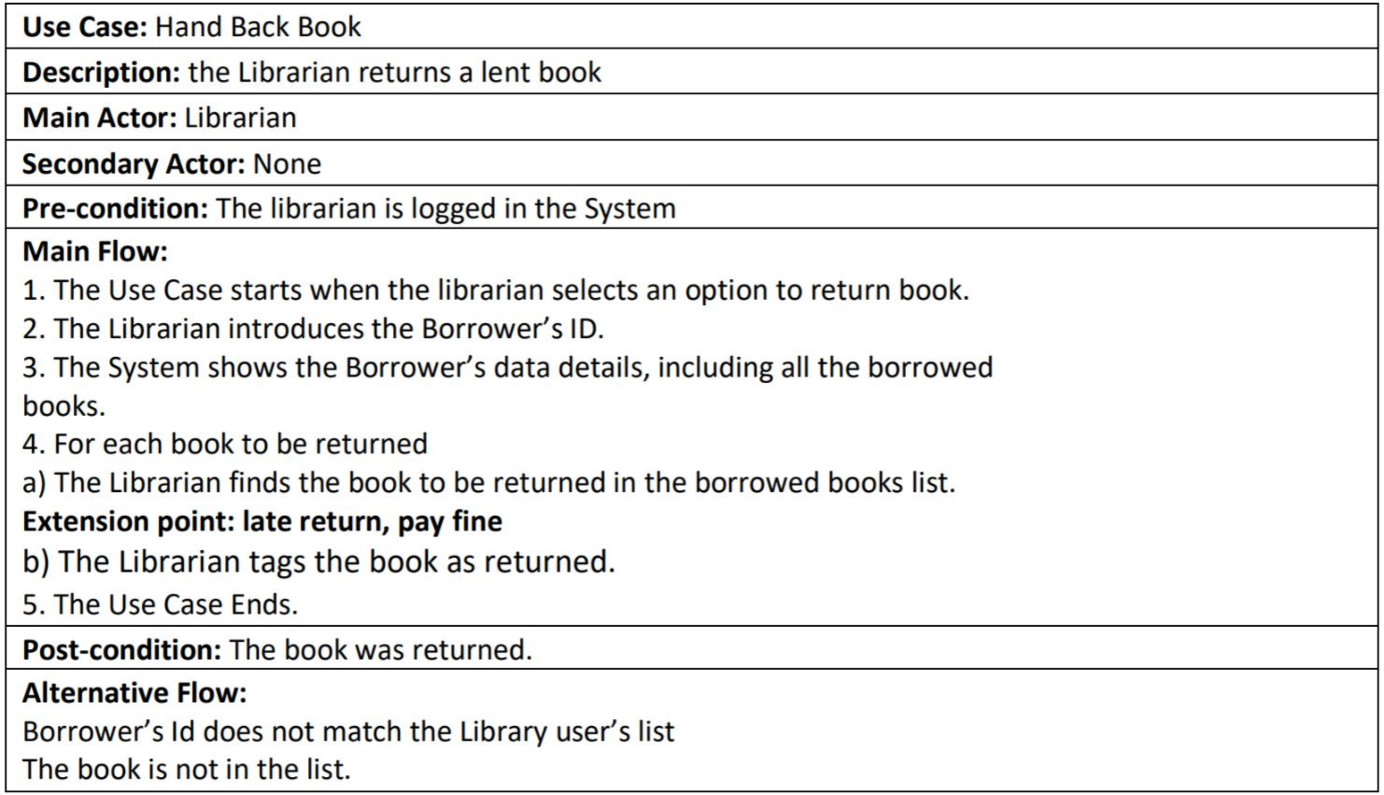
- a) a → b → c → e → d
- b) c → d → a → b → e
- c) a → c → d → b → e
- d) e → a → b → c → d
- e) c → a → e → d → b
- f) a → b → e → d → c



Hand back book

Sequence diagram construction, step by step

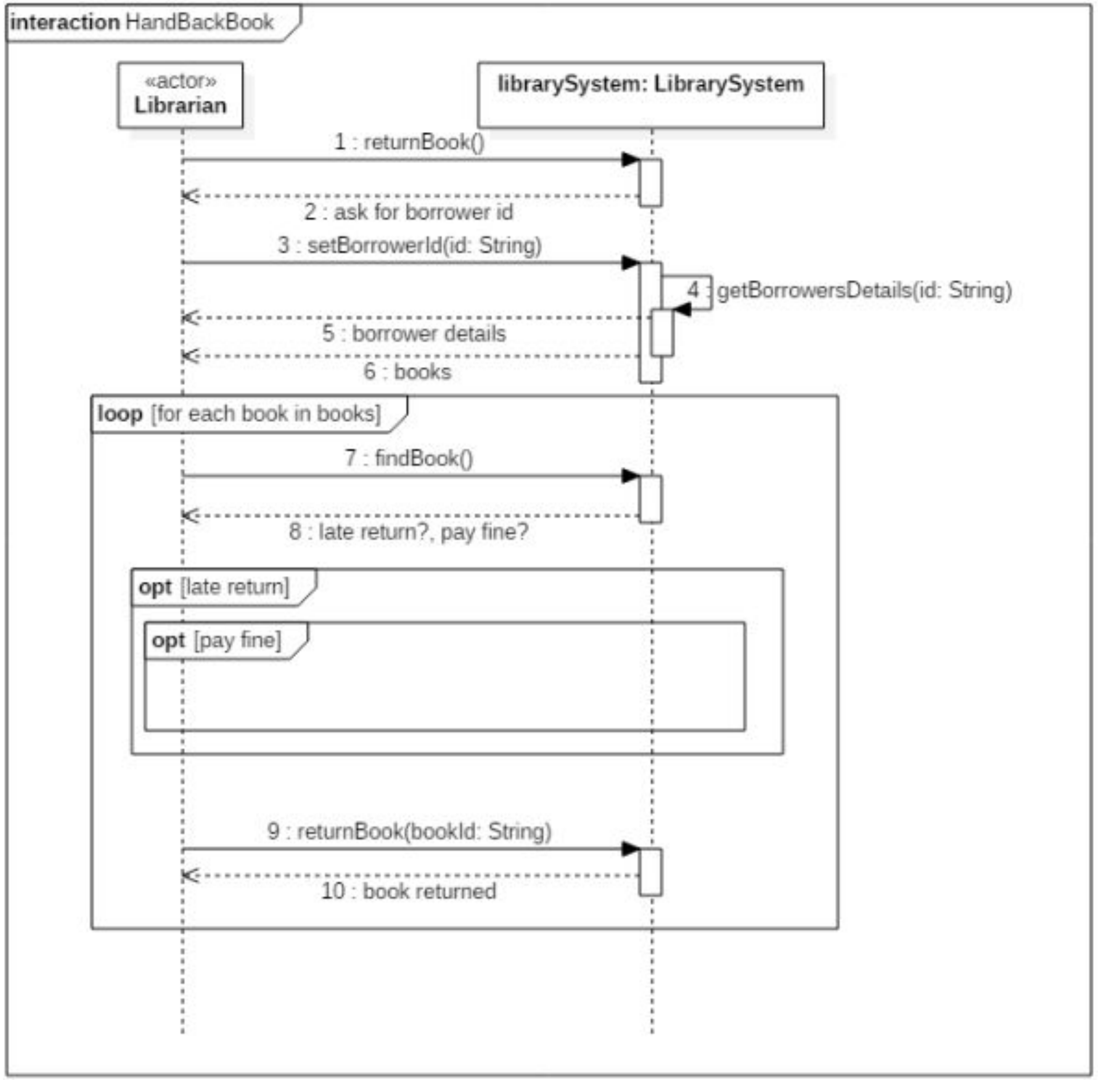
Let us start with the main use case



Use Case: Hand Back Book
Description: the Librarian returns a lent book
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The librarian is logged in the System
Main Flow: 1. The Use Case starts when the librarian selects an option to return book. 2. The Librarian introduces the Borrower's ID. 3. The System shows the Borrower's data details, including all the borrowed books. 4. For each book to be returned a) The Librarian finds the book to be returned in the borrowed books list. **Extension point: late return, pay fine** b) The Librarian tags the book as returned. 5. The Use Case Ends.
Post-condition: The book was returned.
Alternative Flow: Borrower's Id does not match the Library user's list The book is not in the list.

How do we build the corresponding sequence diagram?

- Identify the actor(s)
- Think of the system as a black-box
- Identify all the messages being exchanged from the actor(s) to the system and the system's answers
- Add placeholders for extensions and inclusions, if necessary
  - Leave them blank, for now
- Keep in mind that we will need to add more details, as we progress
  - Start with the main flow
  - Then, add the alternative flows, one by one


interaction HandBackBook
«actor»
Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
3 : setBorrowerId(id: String)
4 : getBorrowersDetails(id: String)
5 : borrower details
6 : books
loop [for each book in books]
7 : findBook()
8 : late return?, pay fine?
opt [late return]
opt [pay fine]
9 : returnBook(bookId: String)
10 : book returned

Start with the Main flow

Use Case: Hand Back Book

Description: the Librarian returns a lent book

Main Actor: Librarian

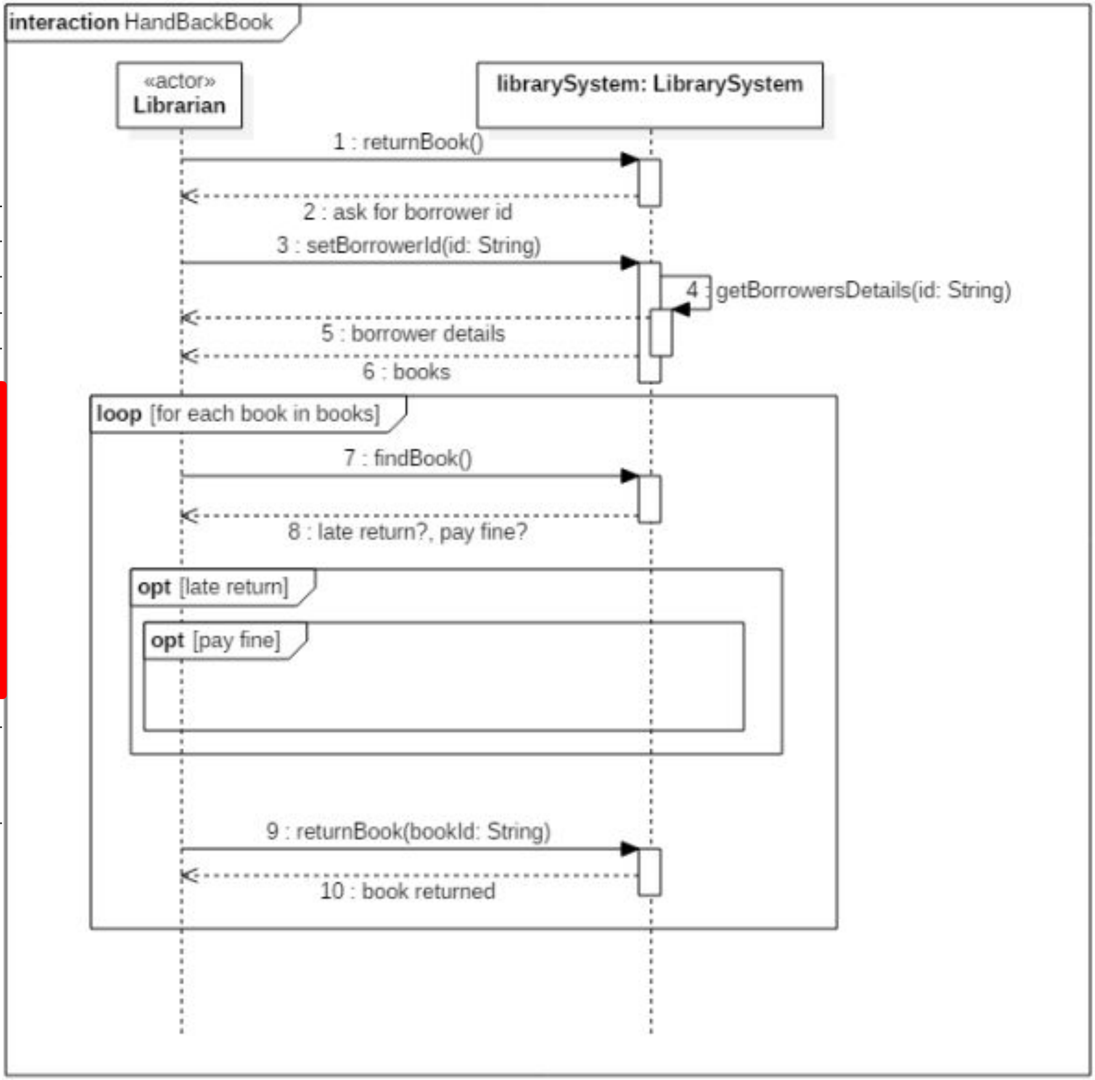
Secondary Actor: None

Pre-condition: The librarian is logged in the System

Main Flow:
1. The Use Case starts when the librarian selects an option to return book.
2. The Librarian introduces the Borrower's ID.
3. The System shows the Borrower's data details, including all the borrowed books.
4. For each book to be returned
a) The Librarian finds the book to be returned in the borrowed books list.
Extension point: late return, pay fine
b) The Librarian tags the book as returned.
5. The Use Case Ends.

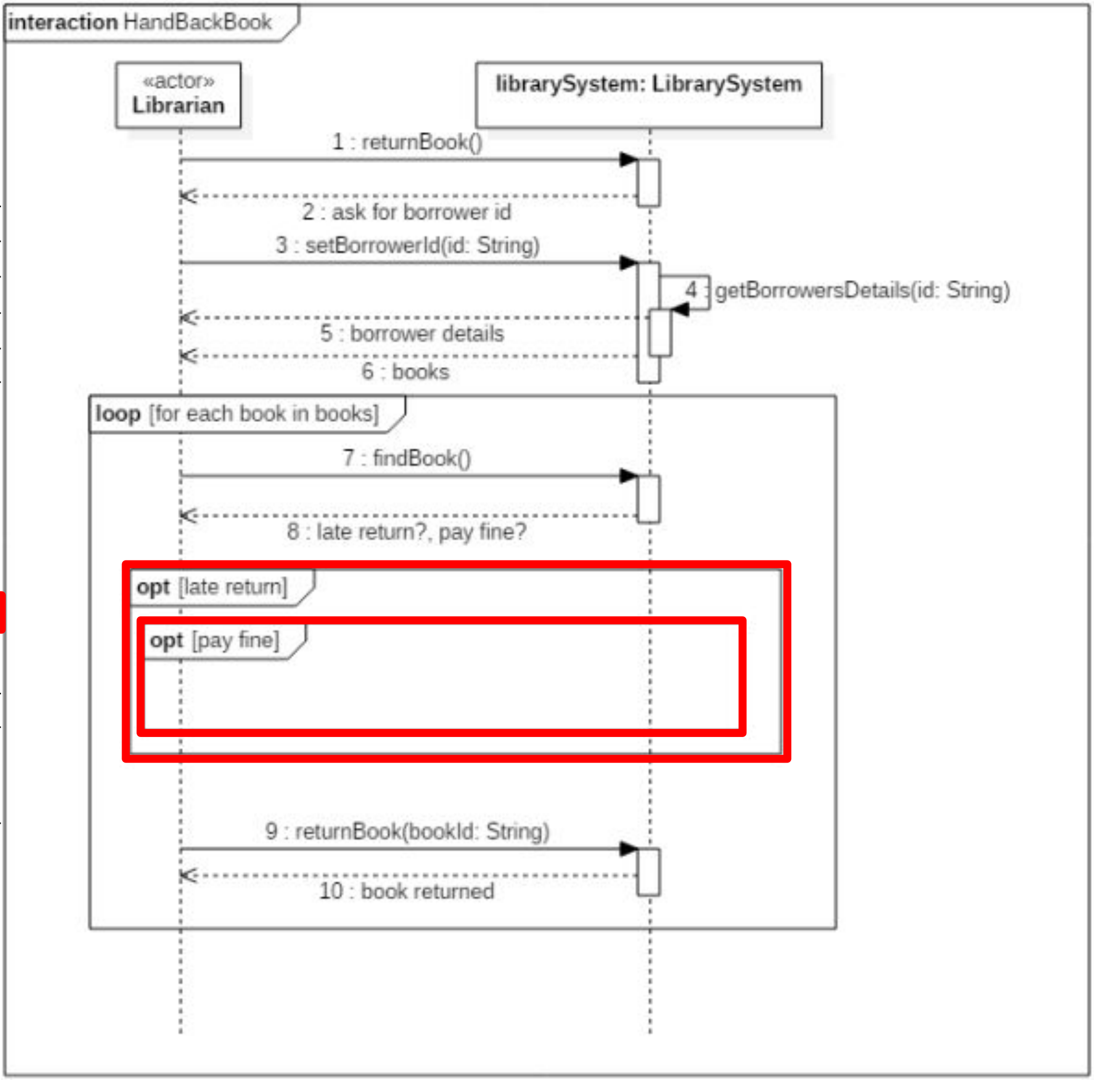
Post-condition: The book was returned.

Alternative Flow:
Borrower's Id does not match the Library user's list
The book is not in the list.


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
3 : setBorrowerId(id: String)
4 : getBorrowersDetails(id: String)
5 : borrower details
6 : books
loop [for each book in books]
7 : findBook()
8 : late return?, pay fine?
opt [late return]
opt [pay fine]
9 : returnBook(bookId: String)
10 : book returned

Extension points execution is optional

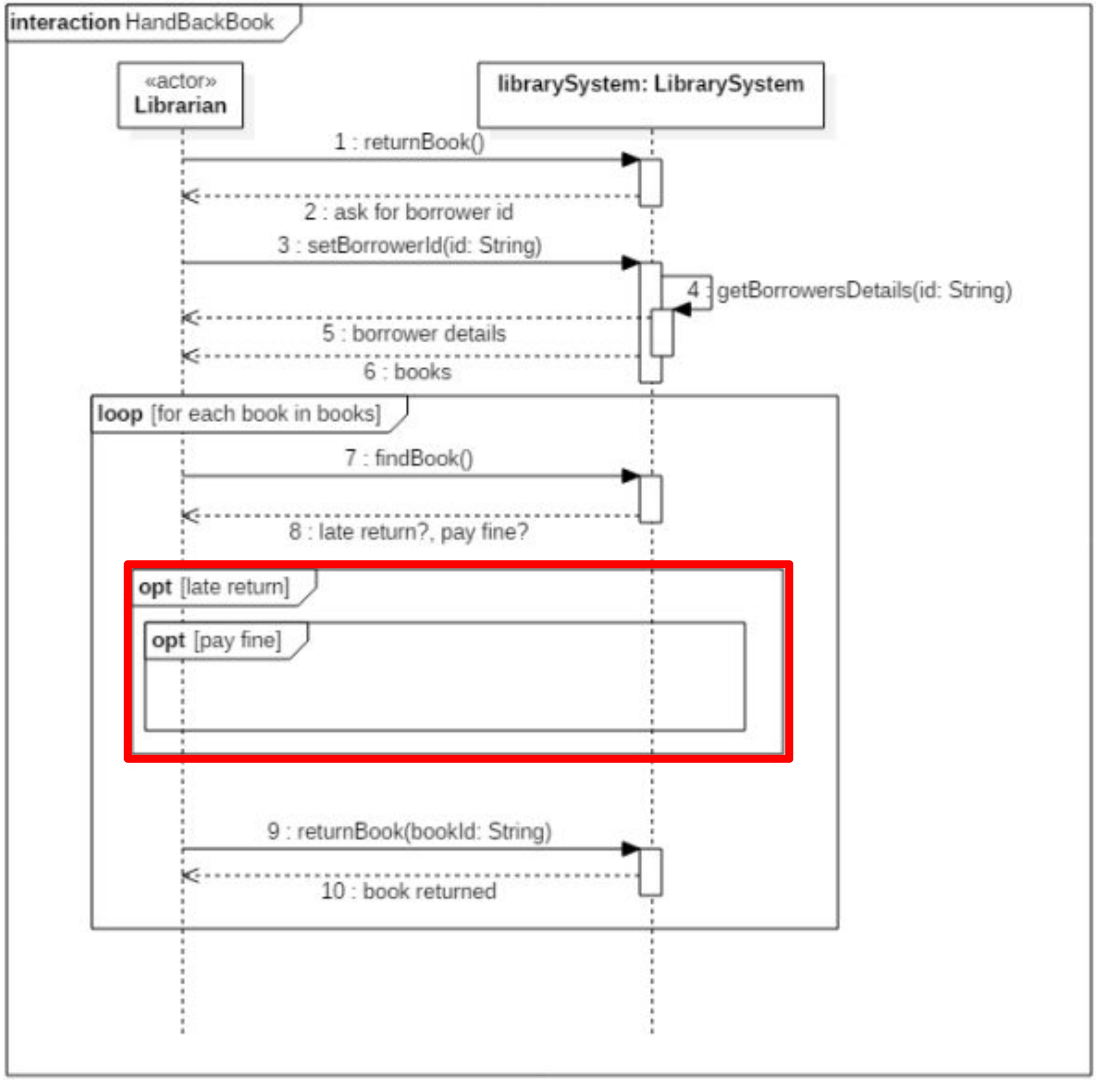
Use Case: Hand Back Book
Description: the Librarian returns a lent book
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The librarian is logged in the System
Main Flow: 1. The Use Case starts when the librarian selects an option to return book. 2. The Librarian introduces the Borrower's ID. 3. The System shows the Borrower's data details, including all the borrowed books. 4. For each book to be returned a) The Librarian finds the book to be returned in the borrowed books list. **Extension point: late return, pay fine** b) The Librarian tags the book as returned. 5. The Use Case Ends.
Post-condition: The book was returned.
Alternative Flow: Borrower's Id does not match the Library user's list The book is not in the list.


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
3 : setBorrowerId(id: String)
4 : getBorrowersDetails(id: String)
5 : borrower details
6 : books
loop [for each book in books]
7 : findBook()
8 : late return?, pay fine?
opt [late return]
opt [pay fine]
9 : returnBook(bookId: String)
10 : book returned

What to do with those late return and pay fine options?

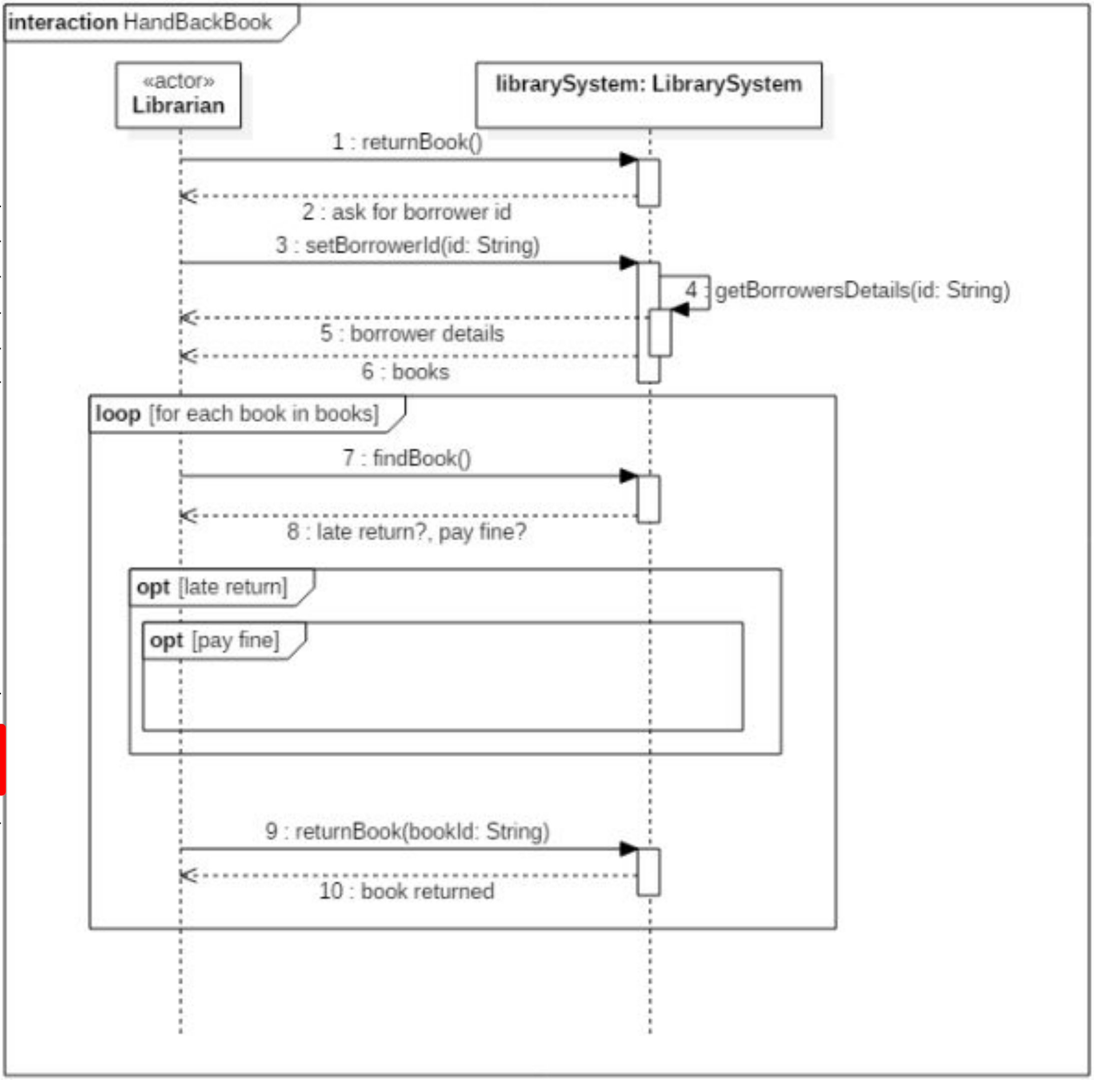
We will have to analyse what happens with the corresponding extension use cases…

For now, leave it like this.

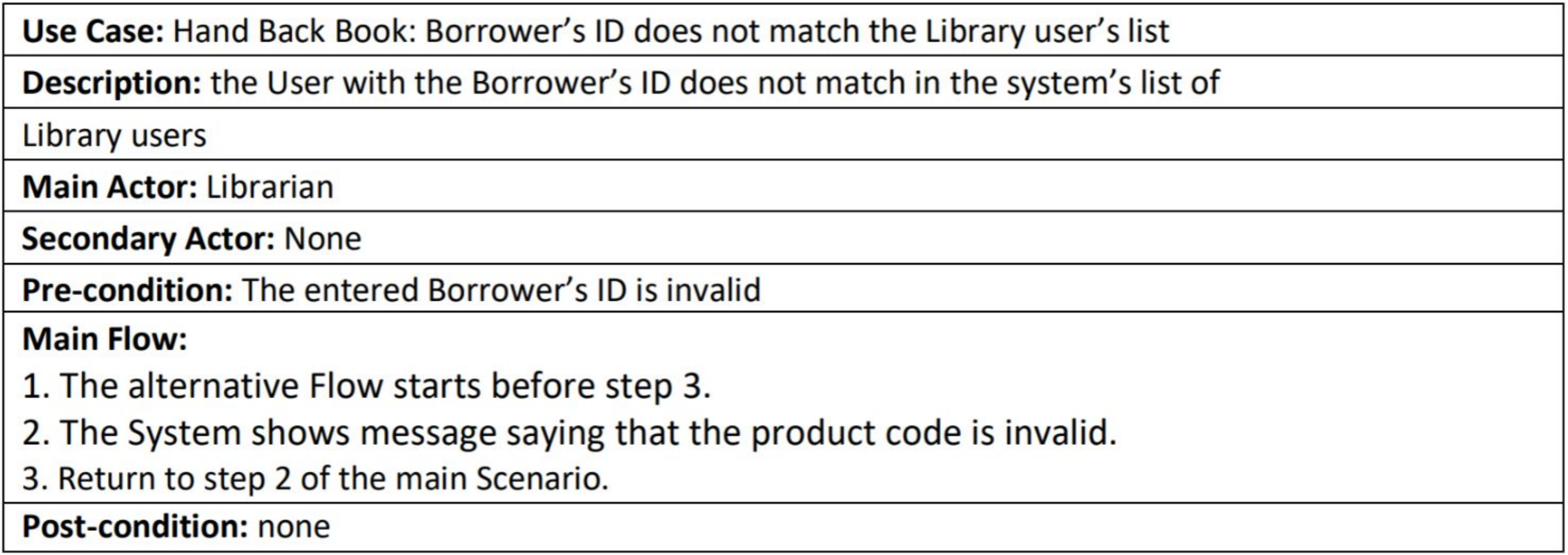

interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
3 : setBorrowerId(id: String)
4 : getBorrowersDetails(id: String)
5 : borrower details
6 : books
loop [for each book in books]
7 : findBook()
8 : late return?, pay fine?
opt [late return]
opt [pay fine]
9 : returnBook(bookId: String)
10 : book returned

Add the first alternative flow

Use Case: Hand Back Book
Description: the Librarian returns a lent book
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The librarian is logged in the System
Main Flow: 1. The Use Case starts when the librarian selects an option to return book. 2. The Librarian introduces the Borrower's ID. 3. The System shows the Borrower's data details, including all the borrowed books. 4. For each book to be returned a) The Librarian finds the book to be returned in the borrowed books list. **Extension point: late return, pay fine** b) The Librarian tags the book as returned. 5. The Use Case Ends.
Post-condition: The book was returned.
Alternative Flow: Borrower's Id does not match the Library user's list The book is not in the list.

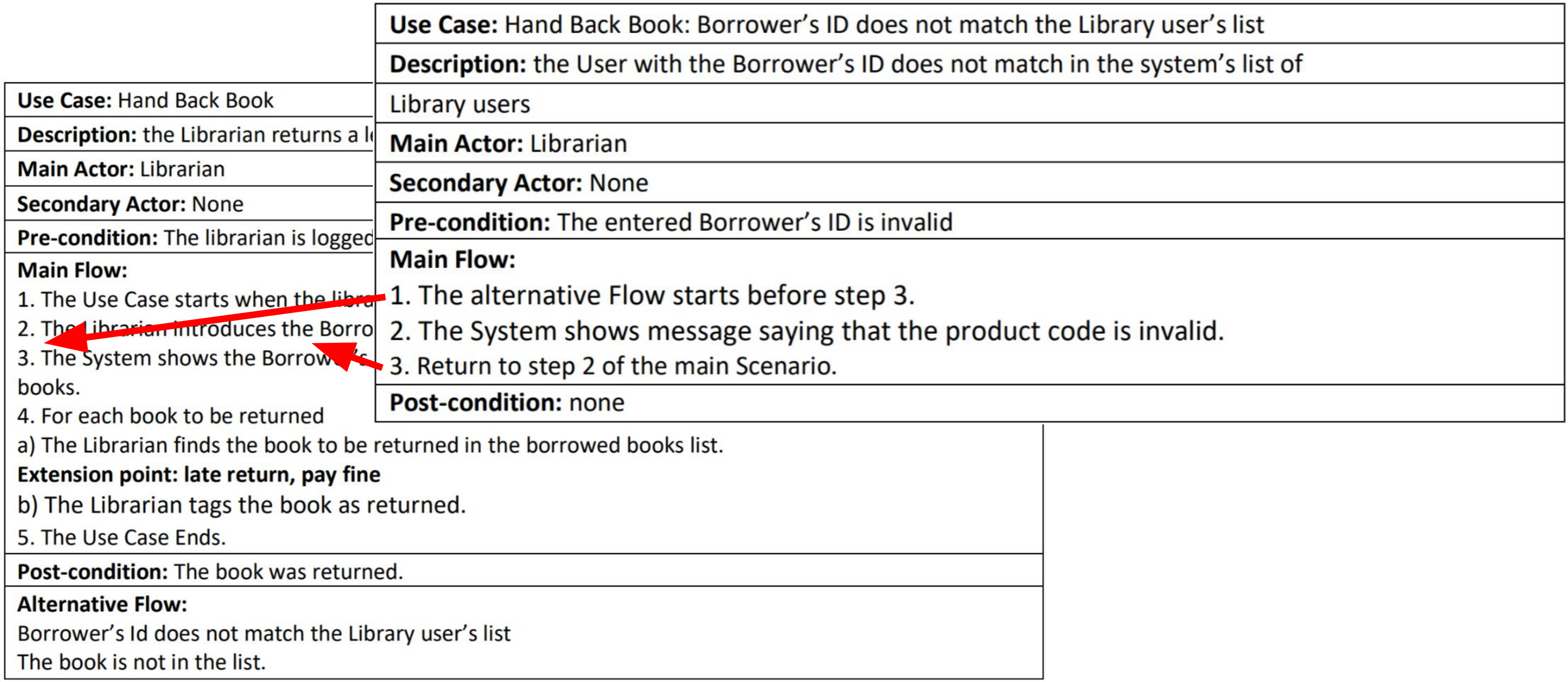

interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
3 : setBorrowerId(id: String)
4 : getBorrowersDetails(id: String)
5 : borrower details
6 : books
loop [for each book in books]
7 : findBook()
8 : late return?, pay fine?
opt [late return]
opt [pay fine]
9 : returnBook(bookId: String)
10 : book returned

Let’s have a look at the alternative flow use case specification



Use Case: Hand Back Book: Borrower’s ID does not match the Library user’s list
Description: the User with the Borrower’s ID does not match in the system’s list of
Library users
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The entered Borrower’s ID is invalid
Main Flow: 1. The alternative Flow starts before step 3. 2. The System shows message saying that the product code is invalid. 3. Return to step 2 of the main Scenario.
Post-condition: none

Find out where the specification fits in the base use case



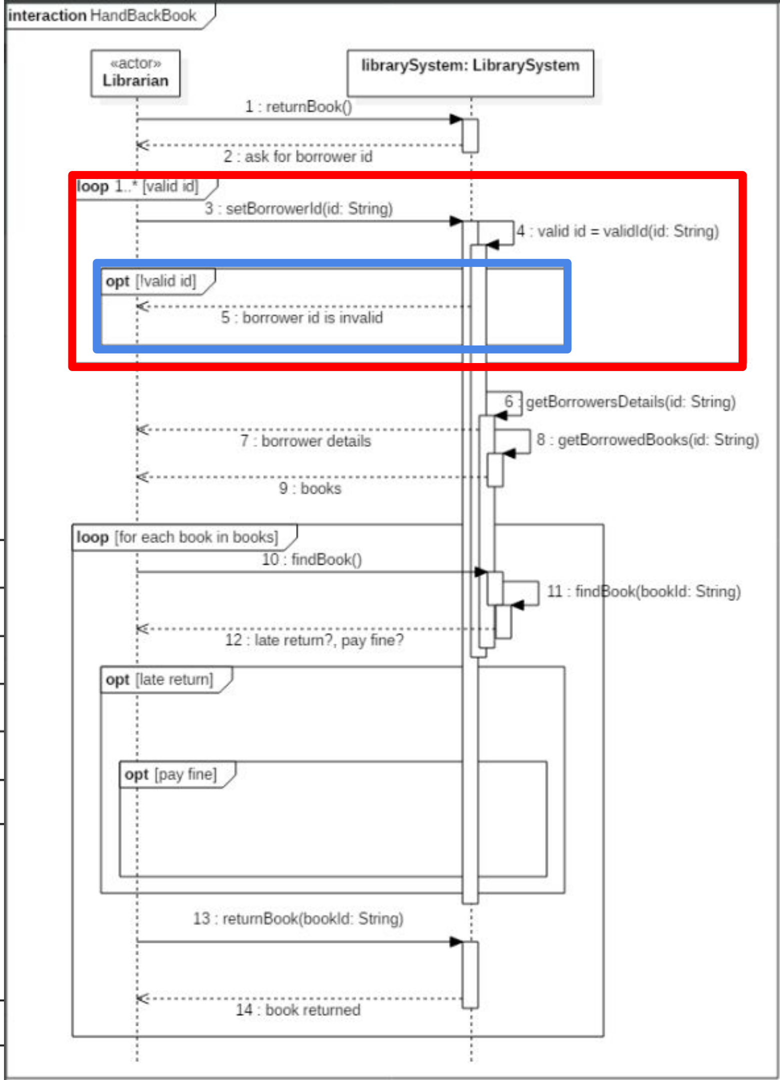
Use Case: Hand Back Book
Description: the Librarian returns a l
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The librarian is logged
Main Flow: 1. The Use Case starts when the libra 2. The Librarian introduces the Borro 3. The System shows the Borrower's books. 4. For each book to be returned a) The Librarian finds the book to be returned in the borrowed books list. **Extension point: late return, pay fine** b) The Librarian tags the book as returned. 5. The Use Case Ends.
Post-condition: The book was returned.
Alternative Flow: Borrower's Id does not match the Library user's list The book is not in the list.

Use Case: Hand Back Book: Borrower's ID does not match the Library user's list
Description: the User with the Borrower's ID does not match in the system's list of
Library users
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The entered Borrower's ID is invalid
Main Flow: 1. The alternative Flow starts before step 3. 2. The System shows message saying that the product code is invalid. 3. Return to step 2 of the main Scenario.
Post-condition: none

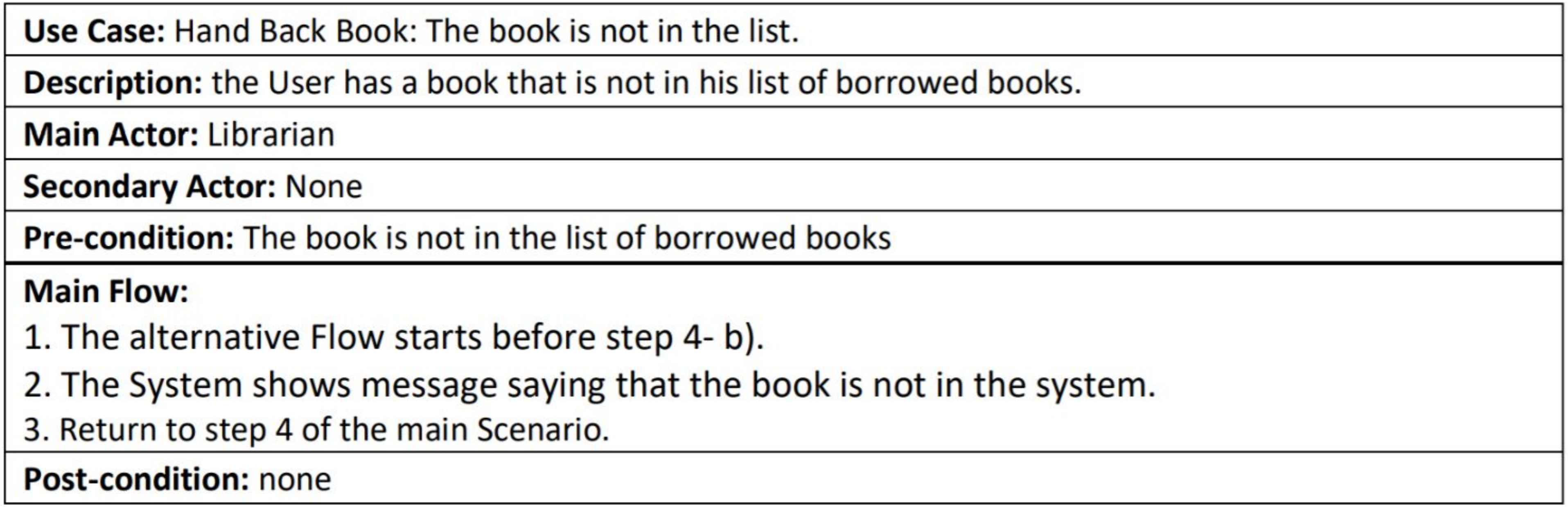
The Librarian repeats the setting of the borrower id until the id is valid

- The loop combined fragment takes care of the repeat until iteration
- The opt combined fragment takes case of the feedback to the librarian

Use Case: Hand Back Book: Borrower's ID does not match the Library user's list
Description: the User with the Borrower's ID does not match in the system's list of
Library users
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The entered Borrower's ID is invalid
Main Flow: 1. The alternative Flow starts before step 3. 2. The System shows message saying that the product code is invalid. 3. Return to step 2 of the main Scenario.
Post-condition: none


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
loop 1..* [valid id]
3 : setBorrowerId(id: String)
4 : valid id = validId(id: String)
opt [!valid id]
5 : borrower id is invalid
6 : getBorrowersDetails(id: String)
7 : borrower details
8 : getBorrowedBooks(id: String)
9 : books
loop [for each book in books]
10 : findBook()
11 : findBook(bookId: String)
12 : late return?, pay fine?
opt [late return]
opt [pay fine]
13 : returnBook(bookId: String)
14 : book returned

What if the book to be returned is not on the list?



Use Case: Hand Back Book: The book is not in the list.
Description: the User has a book that is not in his list of borrowed books.
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The book is not in the list of borrowed books
Main Flow: 1. The alternative Flow starts before step 4- b). 2. The System shows message saying that the book is not in the system. 3. Return to step 4 of the main Scenario.
Post-condition: none

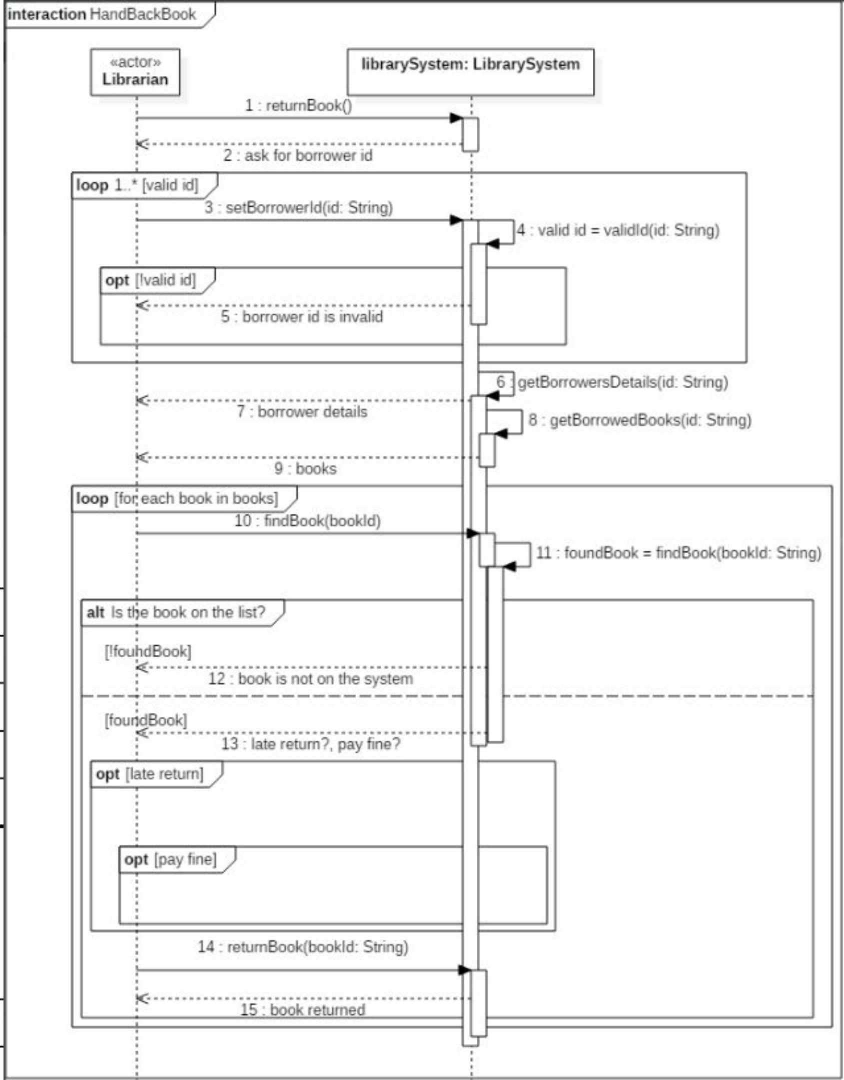
Use Case: Hand Back Book: The book is not in the list.
Description: the User has a book that is not in his list of borrowed books.
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The book is not in the list of borrowed books
Main Flow: 1. The alternative Flow starts before step 4- b). 2. The System shows message saying that the book is not in the system. 3. Return to step 4 of the main Scenario.
Post-condition: none

Use Case: Hand Back Book
Description: the Librarian retu
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The librarian is
Main Flow: 1. The Use Case starts when the librarian selects an option to return book. 2. The Librarian introduces the Borrower's ID. 3. The System shows the Borrower's data details, including all the borrowed books. 4. For each book to be returned a) The Librarian finds the book to be returned in the borrowed books list. **Extension point: late return, pay fine** b) The Librarian tags the book as returned. 5. The Use Case Ends.
Post-condition: The book was returned.
Alternative Flow: Borrower's Id does not match the Library user's list The book is not in the list.

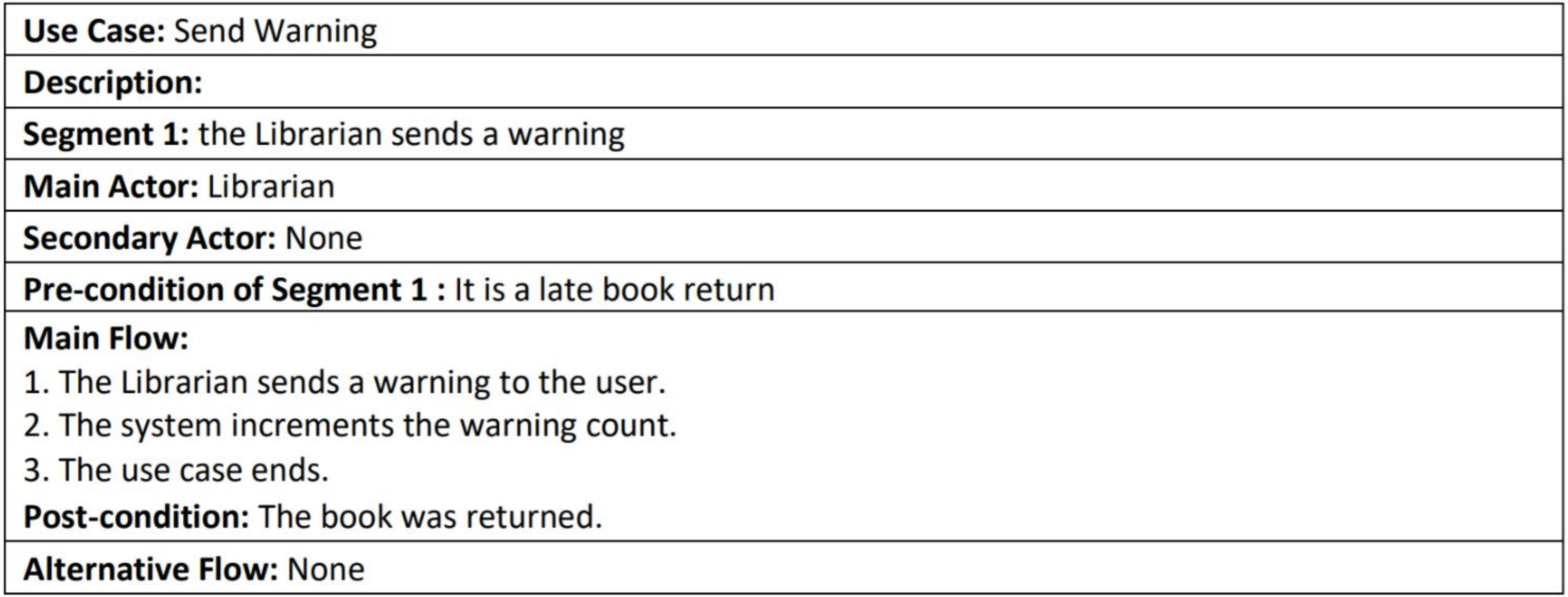
The alt combined fragment supports this alternative

- The !foundBook operand provides the error feedback to the librarian
- The foundBook operand allows returning the book

Use Case: Hand Back Book: The book is not in the list.
Description: the User has a book that is not in his list of borrowed books.
Main Actor: Librarian
Secondary Actor: None
Pre-condition: The book is not in the list of borrowed books
Main Flow: 1. The alternative Flow starts before step 4- b). 2. The System shows message saying that the book is not in the system. 3. Return to step 4 of the main Scenario.
Post-condition: none


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
loop 1..* [valid id]
3 : setBorrowerId(id: String)
4 : valid id = validId(id: String)
opt [!valid id]
5 : borrower id is invalid
6 : getBorrowersDetails(id: String)
7 : borrower details
8 : getBorrowedBooks(id: String)
9 : books
loop [for each book in books]
10 : findBook(bookId)
11 : foundBook = findBook(bookId: String)
alt Is the book on the list?
[!foundBook]
12 : book is not on the system
[foundBook]
13 : late return?, pay fine?
opt [late return]
opt [pay fine]
14 : returnBook(bookId: String)
15 : book returned

If the book is returned late and there is no fine to pay, send a warning

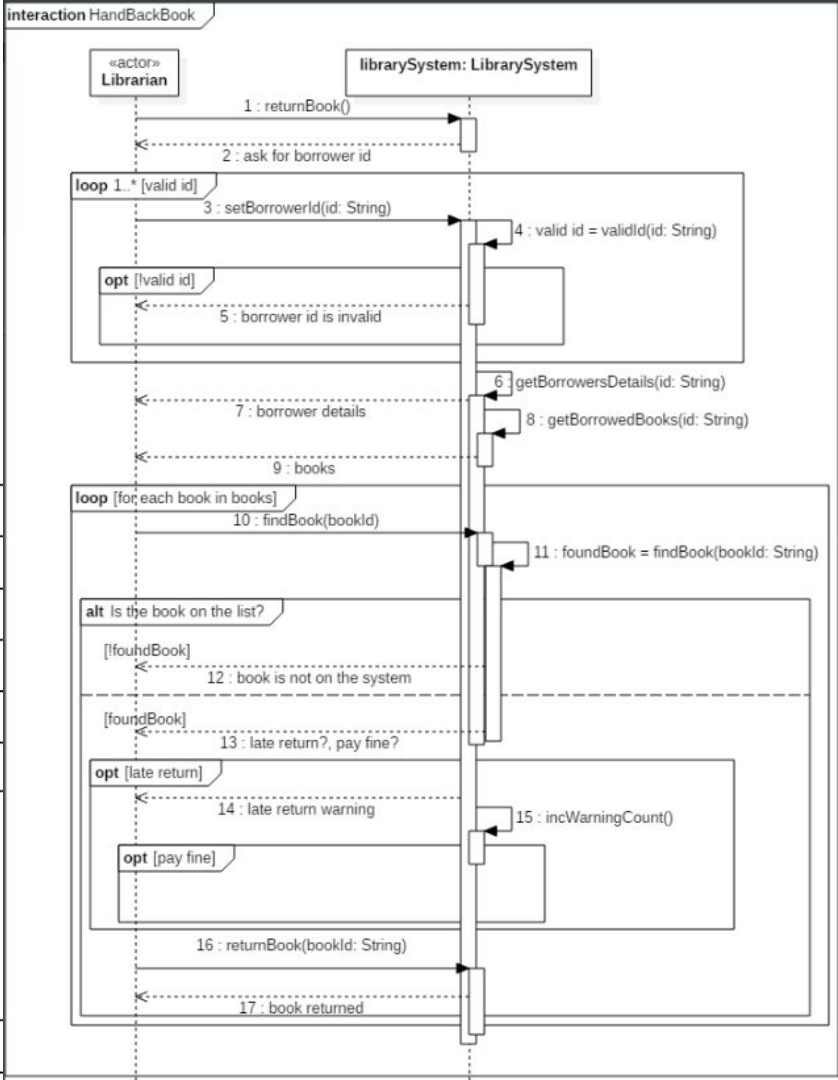


Use Case: Send Warning
Description:
Segment 1: the Librarian sends a warning
Main Actor: Librarian
Secondary Actor: None
Pre-condition of Segment 1 : It is a late book return
Main Flow: 1. The Librarian sends a warning to the user. 2. The system increments the warning count. 3. The use case ends. **Post-condition:** The book was returned.
Alternative Flow: None

We now add the details on what happens when a book is late in the opt combined fragment

- The feedback message to the user
- The increment on the warning counter

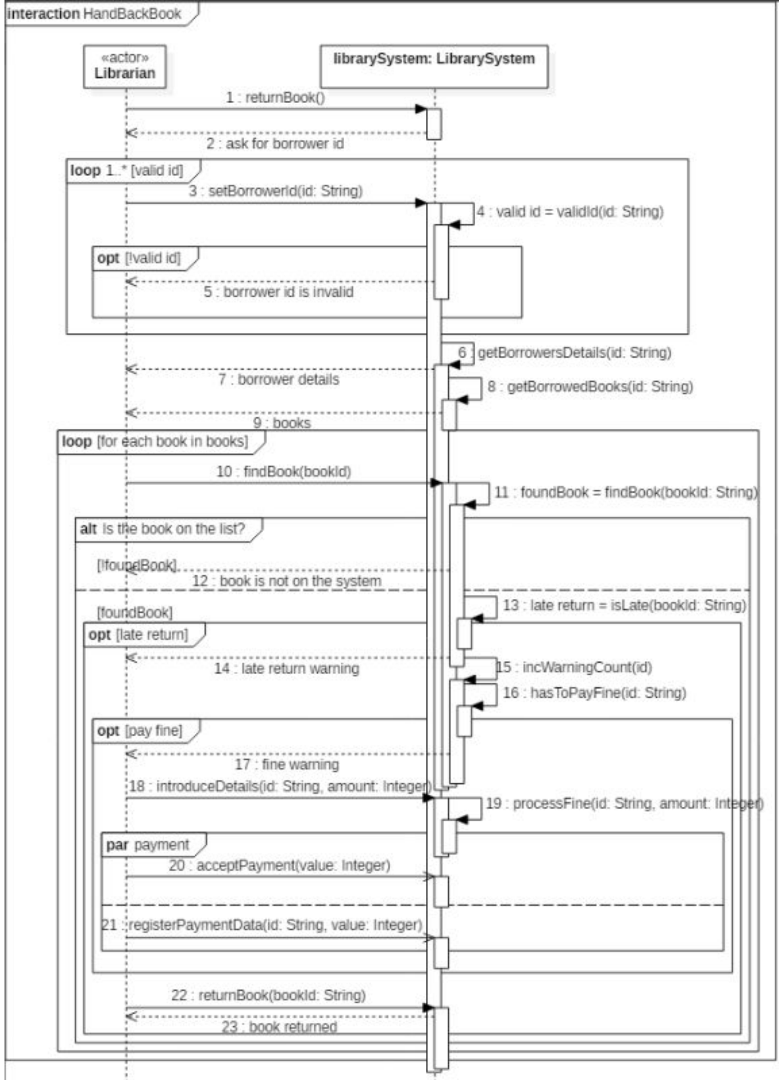
Use Case: Send Warning
Description:
Segment 1: the Librarian sends a warning
Main Actor: Librarian
Secondary Actor: None
Pre-condition of Segment 1 : It is a late book return
Main Flow: 1. The Librarian sends a warning to the user. 2. The system increments the warning count. 3. The use case ends. **Post-condition:** The book was returned.
Alternative Flow: None


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
loop 1..* [valid id]
3 : setBorrowerId(id: String)
4 : valid id = validId(id: String)
opt [!valid id]
5 : borrower id is invalid
6 : getBorrowersDetails(id: String)
7 : borrower details
8 : getBorrowedBooks(id: String)
9 : books
loop [for each book in books]
10 : findBook(bookId)
11 : foundBook = findBook(bookId: String)
alt Is the book on the list?
[!foundBook]
12 : book is not on the system
[foundBook]
13 : late return?, pay fine?
opt [late return]
14 : late return warning
15 : incWarningCount()
opt [pay fine]
16 : returnBook(bookId: String)
17 : book returned

Finally, we fill in the pay fine opt combined fragment

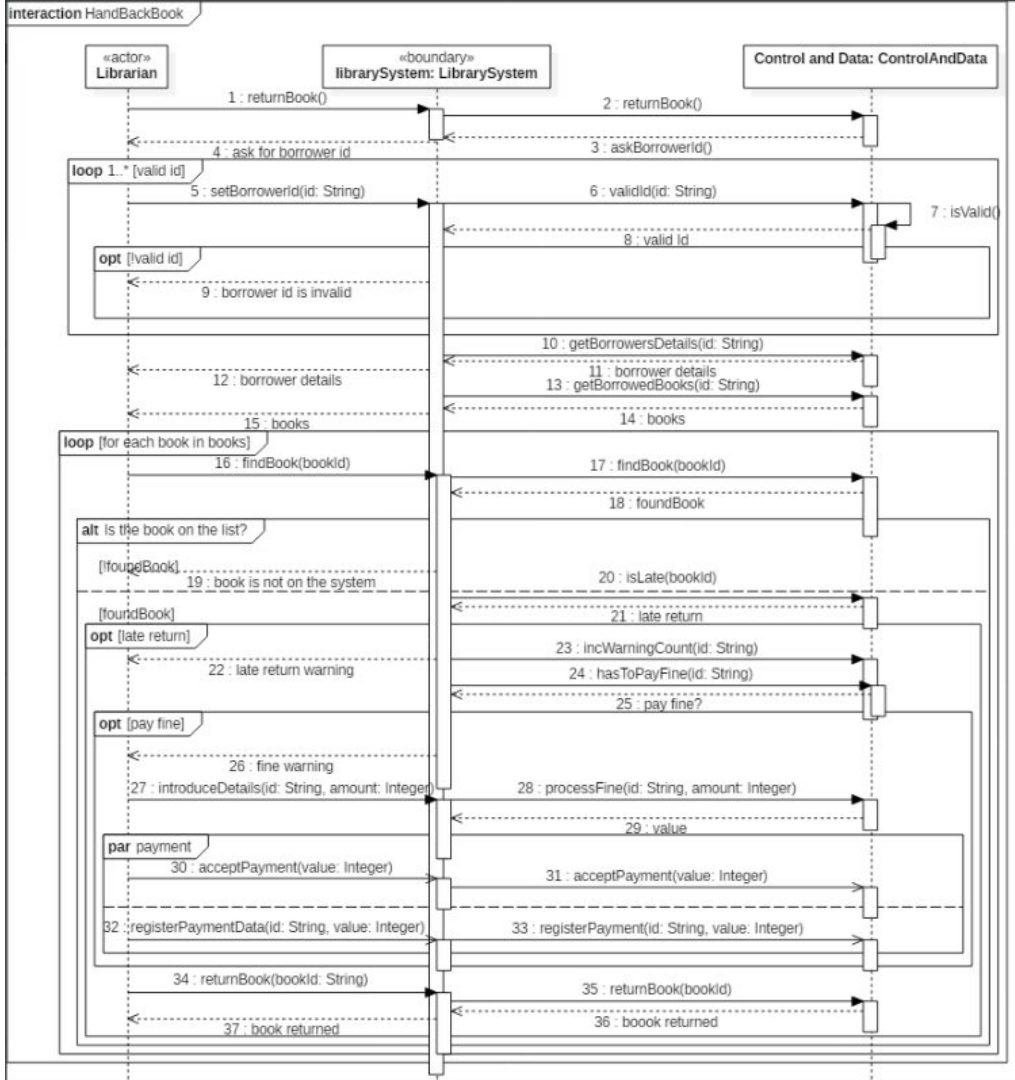
The par combined fragment addresses the two things a librarian can do in parallel:

- Accept payment
- Register payment


interaction HandBackBook
«actor» Librarian
librarySystem: LibrarySystem
1 : returnBook()
2 : ask for borrower id
loop 1..* [valid id]
3 : setBorrowerId(id: String)
4 : valid id = validId(id: String)
opt [!valid id]
5 : borrower id is invalid
6 : getBorrowersDetails(id: String)
7 : borrower details
8 : getBorrowedBooks(id: String)
9 : books
loop [for each book in books]
10 : findBook(bookId)
11 : foundBook = findBook(bookId: String)
alt Is the book on the list?
[!foundBook]
12 : book is not on the system
[foundBook]
13 : late return = isLate(bookId: String)
opt [late return]
14 : late return warning
15 : incWarningCount(id)
16 : hasToPayFine(id: String)
opt [pay fine]
17 : fine warning
18 : introduceDetails(id: String, amount: Integer)
19 : processFine(id: String, amount: Integer)
par payment
20 : acceptPayment(value: Integer)
21 : registerPaymentData(id: String, value: Integer)
22 : returnBook(bookId: String)
23 : book returned

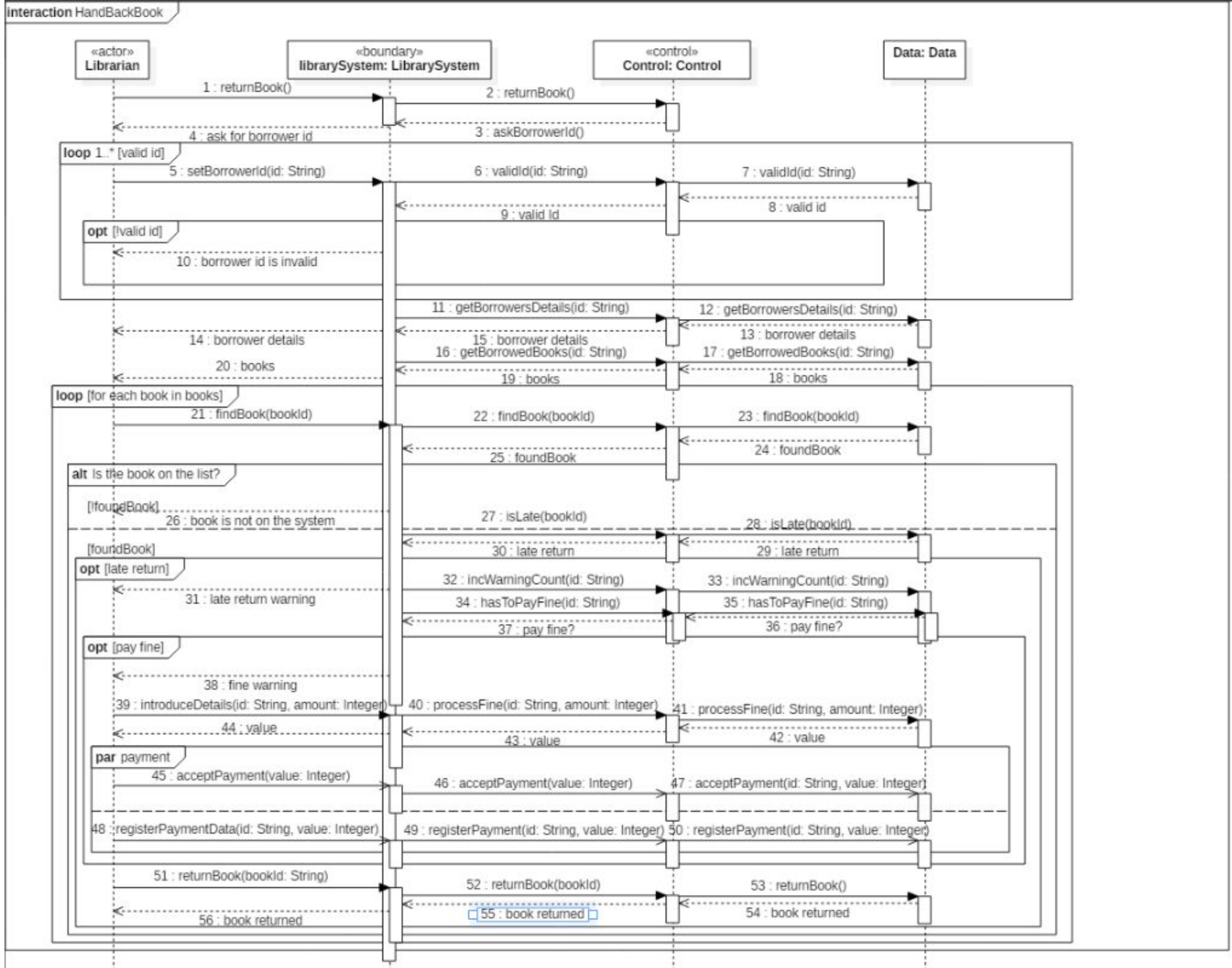
Now, we need to split the LibrarySystem black-box into a boundary classifier and Control & Data

- The actor only interacts with the <<boundary>> classifier
- The <<boundary>> classifier redirects the messages to the system and its replies to the actor


interaction HandBackBook
«actor» Librarian
«boundary» librarySystem: LibrarySystem
Control and Data: ControlAndData
1 : returnBook()
2 : returnBook()
3 : askBorrowerId()
4 : ask for borrower id
loop 1..* [valid id]
5 : setBorrowerId(id: String)
6 : validId(id: String)
7 : isValid()
8 : valid Id
opt [!valid id]
9 : borrower id is invalid
10 : getBorrowersDetails(id: String)
11 : borrower details
12 : borrower details
13 : getBorrowedBooks(id: String)
14 : books
15 : books
loop [for each book in books]
16 : findBook(bookId)
17 : findBook(bookId)
18 : foundBook
alt Is the book on the list?
[!foundBook]
19 : book is not on the system
20 : isLate(bookId)
[foundBook]
21 : late return
opt [late return]
22 : late return warning
23 : incWarningCount(id: String)
24 : hasToPayFine(id: String)
25 : pay fine?
opt [pay fine]
26 : fine warning
27 : introduceDetails(id: String, amount: Integer)
28 : processFine(id: String, amount: Integer)
29 : value
par payment
30 : acceptPayment(value: Integer)
31 : acceptPayment(value: Integer)
32 : registerPaymentData(id: String, value: Integer)
33 : registerPayment(id: String, value: Integer)
34 : returnBook(bookId: String)
35 : returnBook(bookId)
36 : boook returned
37 : book returned

Split control from data

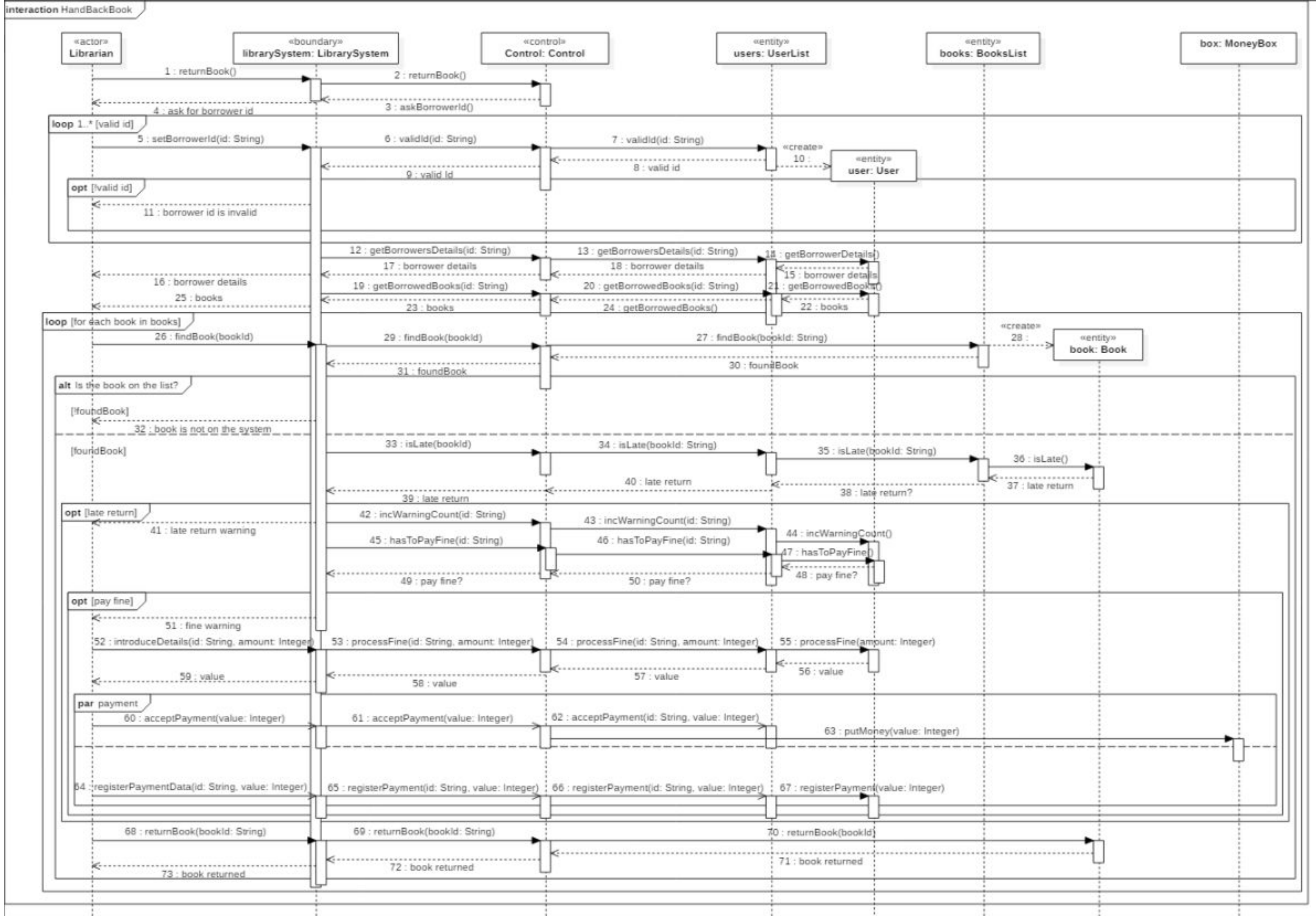
- The <<control>> lifeline orchestrates the interaction
- <<boundary>> communicates with the system via <<control>>



interaction HandBackBook
«actor» Librarian
«boundary» librarySystem: LibrarySystem
«control» Control: Control
Data: Data
1 : returnBook()
2 : returnBook()
3 : askBorrowerId()
4 : ask for borrower id
loop 1..* [valid id]
5 : setBorrowerId(id: String)
6 : validId(id: String)
7 : validId(id: String)
8 : valid id
9 : valid Id
opt [!valid id]
10 : borrower id is invalid
11 : getBorrowersDetails(id: String)
12 : getBorrowersDetails(id: String)
13 : borrower details
14 : borrower details
15 : borrower details
16 : getBorrowedBooks(id: String)
17 : getBorrowedBooks(id: String)
18 : books
19 : books
20 : books
loop [for each book in books]
21 : findBook(bookId)
22 : findBook(bookId)
23 : findBook(bookId)
24 : foundBook
25 : foundBook
alt Is the book on the list?
[!foundBook]
26 : book is not on the system
27 : isLate(bookId)
28 : isLate(bookId)
29 : late return
30 : late return
[foundBook]
opt [late return]
31 : late return warning
32 : incWarningCount(id: String)
33 : incWarningCount(id: String)
34 : hasToPayFine(id: String)
35 : hasToPayFine(id: String)
36 : pay fine?
37 : pay fine?
opt [pay fine]
38 : fine warning
39 : introduceDetails(id: String, amount: Integer)
40 : processFine(id: String, amount: Integer)
41 : processFine(id: String, amount: Integer)
42 : value
43 : value
44 : value
par payment
45 : acceptPayment(value: Integer)
46 : acceptPayment(value: Integer)
47 : acceptPayment(id: String, value: Integer)
48 : registerPaymentData(id: String, value: Integer)
49 : registerPayment(id: String, value: Integer)
50 : registerPayment(id: String, value: Integer)
51 : returnBook(bookId: String)
52 : returnBook(bookId)
53 : returnBook()
54 : book returned
55 : book returned
56 : book returned

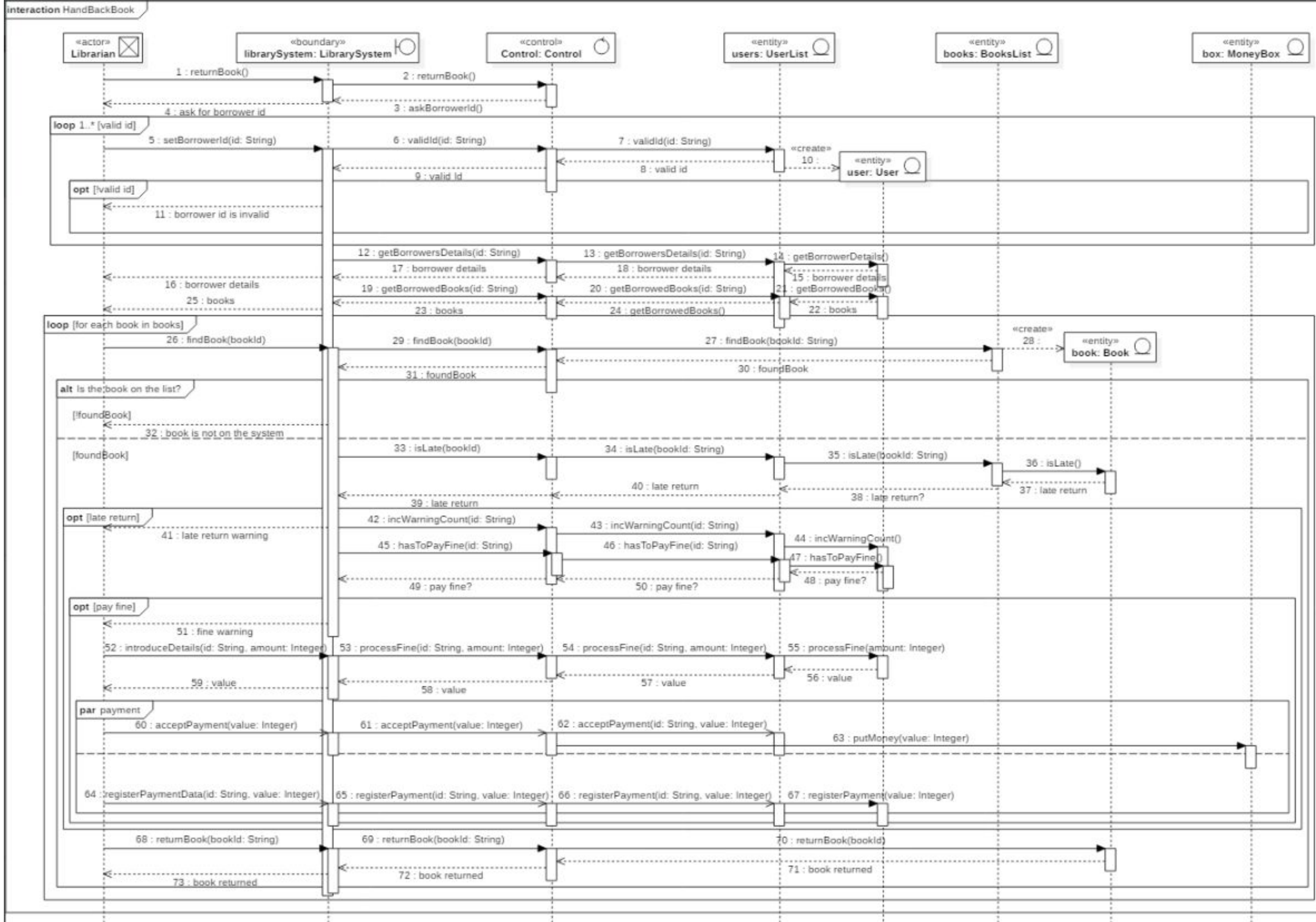
Break data into domain entities

Some of the entities are instantiated during the sequence


interaction HandBackBook
«actor» Librarian
«boundary» librarySystem: LibrarySystem
«control» Control: Control
«entity» users: UserList
«entity» books: BooksList
box: MoneyBox
1 : returnBook()
2 : returnBook()
3 : askBorrowerId()
4 : ask for borrower id
loop 1..* [valid id]
5 : setBorrowerId(id: String)
6 : validId(id: String)
7 : validId(id: String)
«create» 10 :
«entity» user: User
8 : valid id
9 : valid Id
opt [!valid id]
11 : borrower id is invalid
12 : getBorrowersDetails(id: String)
13 : getBorrowersDetails(id: String)
14 : getBorrowerDetails()
15 : borrower details
16 : borrower details
17 : borrower details
18 : borrower details
19 : getBorrowedBooks(id: String)
20 : getBorrowedBooks(id: String)
21 : getBorrowedBooks()
22 : books
23 : books
24 : getBorrowedBooks()
25 : books
loop [for each book in books]
26 : findBook(bookId)
29 : findBook(bookId)
27 : findBook(bookId: String)
«create» 28 :
«entity» book: Book
30 : foundBook
31 : foundBook
alt Is the book on the list?
[!foundBook]
32 : book is not on the system
[foundBook]
33 : isLate(bookId)
34 : isLate(bookId: String)
35 : isLate(bookId: String)
36 : isLate()
37 : late return
38 : late return?
39 : late return
40 : late return
opt [late return]
41 : late return warning
42 : incWarningCount(id: String)
43 : incWarningCount(id: String)
44 : incWarningCount()
45 : hasToPayFine(id: String)
46 : hasToPayFine(id: String)
47 : hasToPayFine()
48 : pay fine?
49 : pay fine?
50 : pay fine?
opt [pay fine]
51 : fine warning
52 : introduceDetails(id: String, amount: Integer)
53 : processFine(id: String, amount: Integer)
54 : processFine(id: String, amount: Integer)
55 : processFine(amount: Integer)
56 : value
57 : value
58 : value
59 : value
par payment
60 : acceptPayment(value: Integer)
61 : acceptPayment(value: Integer)
62 : acceptPayment(id: String, value: Integer)
63 : putMoney(value: Integer)
64 : registerPaymentData(id: String, value: Integer)
65 : registerPayment(id: String, value: Integer)
66 : registerPayment(id: String, value: Integer)
67 : registerPayment(value: Integer)
68 : returnBook(bookId: String)
69 : returnBook(bookId: String)
70 : returnBook(bookId)
71 : book returned
72 : book returned
73 : book returned

Decorate the lifelines

This makes it easier to read the diagram



interaction HandBackBook
«actor» Librarian
«boundary» librarySystem: LibrarySystem
«control» Control: Control
«entity» users: UserList
«entity» books: BooksList
«entity» box: MoneyBox
1 : returnBook()
2 : returnBook()
3 : askBorrowerId()
4 : ask for borrower id
loop 1..* [valid id]
5 : setBorrowerId(id: String)
6 : validId(id: String)
7 : validId(id: String)
«create» 10 :
«entity» user: User
8 : valid id
9 : valid Id
opt [!valid id]
11 : borrower id is invalid
12 : getBorrowersDetails(id: String)
13 : getBorrowersDetails(id: String)
14 : getBorrowerDetails()
15 : borrower details
17 : borrower details
18 : borrower details
16 : borrower details
19 : getBorrowedBooks(id: String)
20 : getBorrowedBooks(id: String)
21 : getBorrowedBooks()
25 : books
23 : books
24 : getBorrowedBooks()
22 : books
loop [for each book in books]
26 : findBook(bookId)
29 : findBook(bookId)
27 : findBook(bookId: String)
«create» 28 :
«entity» book: Book
30 : foundBook
31 : foundBook
alt Is the book on the list?
[!foundBook]
32 : book is not on the system
[foundBook]
33 : isLate(bookId)
34 : isLate(bookId: String)
35 : isLate(bookId: String)
36 : isLate()
37 : late return
38 : late return?
40 : late return
39 : late return
opt [late return]
41 : late return warning
42 : incWarningCount(id: String)
43 : incWarningCount(id: String)
44 : incWarningCount()
45 : hasToPayFine(id: String)
46 : hasToPayFine(id: String)
47 : hasToPayFine()
48 : pay fine?
49 : pay fine?
50 : pay fine?
opt [pay fine]
51 : fine warning
52 : introduceDetails(id: String, amount: Integer)
53 : processFine(id: String, amount: Integer)
54 : processFine(id: String, amount: Integer)
55 : processFine(amount: Integer)
56 : value
57 : value
58 : value
59 : value
par payment
60 : acceptPayment(value: Integer)
61 : acceptPayment(value: Integer)
62 : acceptPayment(id: String, value: Integer)
63 : putMoney(value: Integer)
64 : registerPaymentData(id: String, value: Integer)
65 : registerPayment(id: String, value: Integer)
66 : registerPayment(id: String, value: Integer)
67 : registerPayment(value: Integer)
68 : returnBook(bookId: String)
69 : returnBook(bookId: String)
70 : returnBook(bookId)
71 : book returned
72 : book returned
73 : book returned

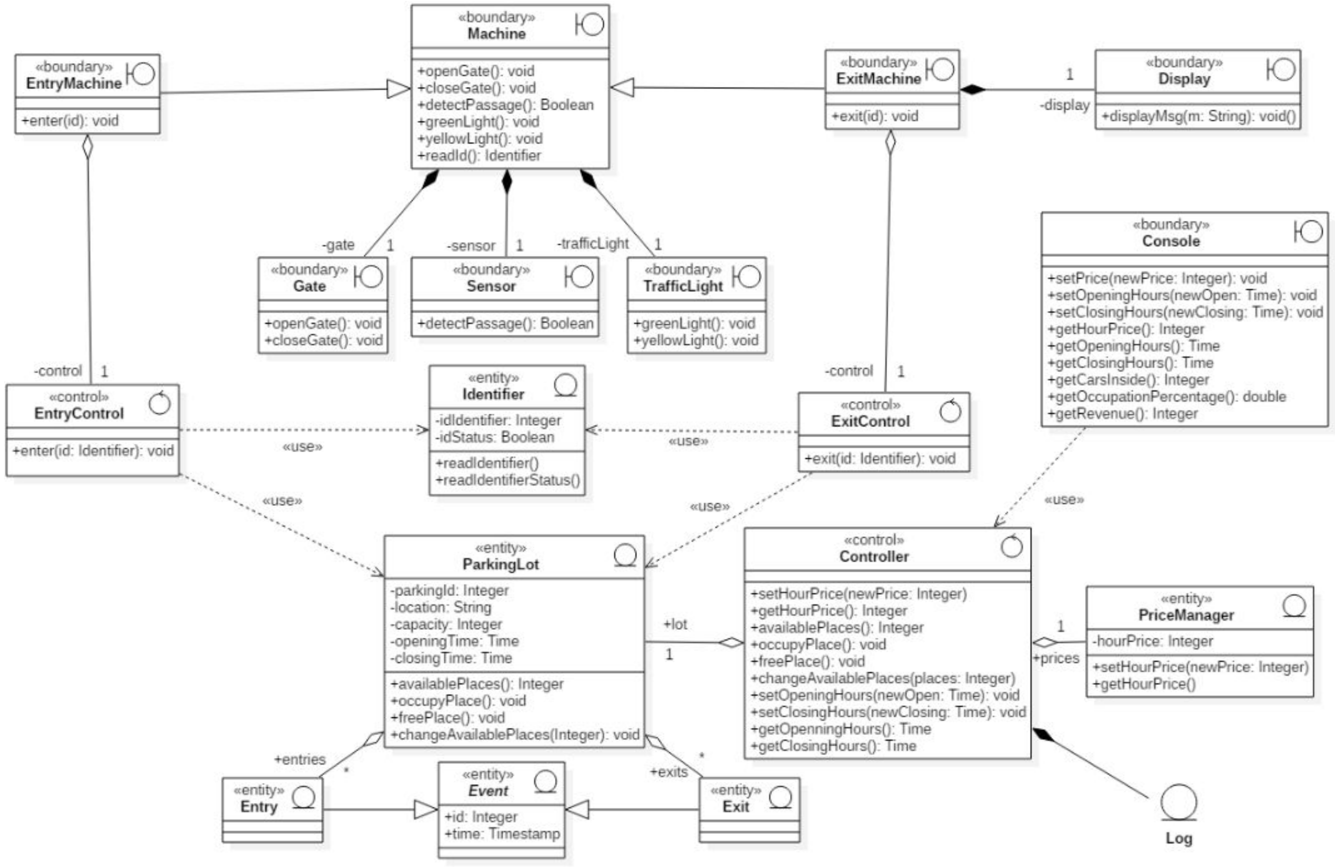
Some takeaways

- These diagrams should be built stepwise
- Actors only exchange messages with <<boundary>> lifelines
- <<boundary>> lifelines pass those messages on to <<control>> lifelines
- <<control>> lifelines orchestrate the remaining interactions with domain objects
- In this particular case, we did not split neither the <<boundary>> lifeline nor the <<control>> lifeline, although this often happens in other sequence diagrams

Package and Component Diagrams

Create a package and a component diagram from a class diagram

Consider the following class diagram fragment



When devising your solutions

- Remember to build a layered architecture (3 levels should do)
- On the package diagram
    - Remember that top levels are aware of the level right below them, but should not be aware of lower levels than that
- On the component diagram, a first approach can be to evolve from the packages. Then, the next hint is to be inspired by physical devices.
    - In other words, you may start with 3 mega-components and then break them down into finer grained components
    - Add interfaces as necessary, to create a component assembly for your architecture