

Software Engineering 2 Assignment

Ian Hipolito – C21436494

Extend and Test a More Comprehensive USE Model for The Library System In USE

Introduction:

I have chosen option 1 (extend and test a more comprehensive USE model for the library system in USE) out of all the options we were given for this assignment. I have created multiple new use cases to extend and develop the library system. One use case that I have implemented in this system is a reservation feature. This allows members to reserve or unreserve copies of a book. I have also created a member status management use case that allows you to change or update a member's status such "active", "inactive" and "expired". Based on a member's status they will or will not be able to borrow, return, reserve or unreserve books or copies, etc. I have also created a use case that allows the user and member to set a rating on a book.

Reserve Use Case:

In the reserve use case, for a member to reserve a copy, the code checks if the member meets the eligible/correct requirements to be able to reserve a copy of a book. This is where is "okToReserve" operation comes in and checks the eligibility of a member. Examples of these requirements to reserve a copy of a book, that the "okToReserve" operation verifies are that a member must have less than two books on loan and an active status (cannot have an inactive or expired status), etc. When a member reserves a copy of a book, the reservation is recorded, and their reserved copy count is increased. The copy status is also set to "#onReserve". This also correlates to the corresponding code in the "Copy" class, as the copy's reserved count is incremented, and the "reserve" operation is also called in the associated Book object. In the "Book" class, the reserved count on the shelf is increased, and the available count is decreased respectively. However, if there are already two copies of a book reserved, then the book's status would be set to "#unavailable" and the member will not be able to reserve that copy of a book.

Unreserve Use Case:

The unreserve use case is an operation defined and used in all three classes, "Book", "Copy" and "Member" classes, similar to the reserve operation. In the "Copy" class, the unreserve operation is used to remove the reservation of a specific copy of a book and updates its status to "OnShelf", decreases the number of copies on reserve "no_onshef_reserved" and increases the count of copies available on the shelf "no_onshef". In the "Book" class, when the operation is called, it decreases the count of reserved copies and increases the count of

available copies, while also updating the book's status to "#available". In the "Member" class, the unreserve operation removes the copy from the member's reserved list, updates all the relevant counts and calls "unreserve()" in the "Copy" class. Overall, the unreserve use case is responsible for cancelling a reservation of a copy of a book.

Member Status Management Use Cases:

I implemented a member status management use case in the library system to regulate the privileges that members have depending on their status. Members can have three different statuses, "active", "inactive" and "expired", which then determines their access to different library resources such as being able to reserve or borrow a book, etc. You can change a member's status by using these operations, "active()", "inactive()" and "expired()". This system ensures that only active members can reserve or borrow copies and books. If a member's status is updated to be inactive or expired, they will not be able to reserve or borrow and are restricted until their status is updated again to be active.

SetRating & SetBookRating Use cases:

These two use cases were created/ developed so that the user or members may rate a book. The "setRating" action of the "Book" class changes a book's rating to the integer value that a user or member has entered. Users may only enter ratings that are larger than 0 or less than 10, and if they enter an integer outside of these ranges, the rating won't be changed. The "setBookRating" action of the "Member" class enables a member to assign their own rating to a book. The operation verifies whether a member has previously borrowed a copy of the related book before allowing them to submit a rating. After the member has set a rating on a book, it updates the book's rating using the "setRating" operation defined in the "Book" class.

Conclusion:

Overall, these use cases that I have created and implemented extend and enhance the library system. The reserve use case provides the members to reserve a copy of a book effectively and efficiently, while efficiently managing the availability of copies and prevents overbooking by the members. In relation to the reserve use case, the unreserve use case allows members to cancel their bookings, while efficiently updating the counts and statuses accurately. The member management system use case is effective as it helps maintain control over a member's privileges and restrictions depending on their statuses. This ensures that only active members can borrow, reserve a copy of a book, etc. Finally, the setRating and setBookRating use cases allow members to provide ratings and gather information on the books the members have borrowed. These use cases largely contribute to the effectiveness and efficiency of the library system.

The USE Code I Have Written:

```
model Library

enum BookStatus { available, unavailable, onreserve}
enum CopyStatus { onShelf, onLoan, onReserve }
enum MemberStatus { active, inactive, expired }

class Book
  attributes
    title : String
    author : String
    status : BookStatus init = #available
    no_copies : Integer init = 2
    no_onshelf : Integer init = 2
    no_onshelf_reserved : Integer init = 0
    rating : Integer init = 0

  operations
    borrow()
    begin
      self.no_onshelf := self.no_onshelf - 1;
      if (self.no_onshelf = 0) then
        self.status := #unavailable
      end
    end

    reserve()
    begin
      self.no_onshelf_reserved := self.no_onshelf_reserved + 1;
      self.no_onshelf := self.no_onshelf - 1;
      if (self.no_onshelf_reserved >= 2 or self.no_onshelf = 0) then
        self.status := #unavailable
      end
    end

    unreserve()
    begin
      self.no_onshelf_reserved := self.no_onshelf_reserved - 1;
      self.no_onshelf := self.no_onshelf + 1;
      self.status := #available
    end

    return() begin
      self.no_onshelf := self.no_onshelf + 1;
      self.status := #available
    end

    setRating(r : Integer)
```

```

        begin
            self.rating := r
        end

    statemachines
        psm States
        states
            newTitle : initial
            available      [no_onshef > 0]
            unavailable    [no_onshef = 0]
        transitions
            newTitle -> available { create }
            available -> unavailable { [no_onshef = 1] borrow() }
            available -> available { [no_onshef > 1] borrow() }
            available -> unavailable { [no_onshef = 1] reserve() }
            available -> available { [no_onshef > 1] reserve() }
            available -> available { return() }
            unavailable -> available { return() }
        end
    end
end

class Copy
    attributes
        status : CopyStatus init = #onShelf
        no_onshef : Integer init = 1
        no_onshef_reserved : Integer init = 0
        book : Book

    operations
        return()
        begin
            self.no_onshef := self.no_onshef + 1;
            self.status := #onShelf;
            self.book.return()
        end
        pre: self.status = #onLoan
        post: no_onshef = no_onshef@pre + 1

        reserve(m : Member)
        begin
            self.no_onshef_reserved := self.no_onshef_reserved + 1;
            self.status := #onReserve;
            self.book.reserve()
        end
        pre: self.status = #onShelf

        unreserve()
        begin

```

```

        self.no_onshef_reserved := self.no_onshef_reserved - 1;
        self.status:= #onShelf;
        self.book.unreserve()
    end
    pre: self.status = #onReserve
    post: no_onshef_reserved = no_onshef_reserved@pre - 1

    borrow(m : Member)
    begin
        if (self.no_onshef = 0) then
            self.status:= #onLoan
        else
            self.no_onshef := self.no_onshef - 1;
            self.status:= #onLoan;
            self.book.borrow()
        end
    end
end

statemachines
    psm States
    states
        newCopy : initial
        onShelf      [no_onshef > 0]
        onLoan       [no_onshef = 0]
        onReserve    [no_onshef_reserved > 0]
    transitions
        newCopy -> onShelf { create }
        onShelf -> onLoan { borrow() }
        onLoan -> onShelf { return() }
        onShelf -> onReserve { reserve() }
        onReserve -> onShelf { unreserve() }
    end
end

class Member
    attributes
        name : String
        address : String
        no_onloan : Integer init = 0
        no_reserved : Integer init = 0
        status : MemberStatus init = #active
        fine : Integer
    operations
        okToBorrow() : Boolean
    begin
        if (self.no_onloan < 2 and self.status = #active) then
            result := true
        else

```

```

        result := false
    end
end

okToReserve() : Boolean
begin
    if (self.no_onloan < 2 and self.status = #active) then
        result := true
    else
        result := false
    end
end

borrow(c : Copy)
begin
    declare ok : Boolean;
    ok := self.okToBorrow();
    if( ok ) then
        insert (self, c) into HasBorrowed;
        self.no_onloan := self.no_onloan + 1;

        c.borrow(self)
    end
end

reserve(c : Copy)
begin
    declare ok : Boolean;
    ok := self.okToReserve();
    if( ok ) then
        insert (self, c) into HasReserved;
        self.no_reserved := self.no_reserved + 1;
        c.reserve(self)
    end
end

unreserve(c : Copy)
begin
    delete (self, c) from HasReserved;
    self.no_reserved := self.no_reserved - 1;
    c.unreserve()
end

return(c : Copy)
begin
    delete (self, c) from HasBorrowed;
    self.no_onloan := self.no_onloan - 1;
    c.return()
end

```

```

end

setBookRating(b: Book, r: Integer)
begin
    if (self.borrowed->exists(c | c.book = b)) then
        b.setRating(r)
    end
end

active()
begin
    self.status := #active
end

inactive()
begin
    self.status := #inactive
end

expired()
begin
    self.status := #expired
end

statemachines
    psm States
    states
        newMember : initial
        active
        inactive
        expired
    transitions
        newMember -> active { create }
        active -> inactive { inactive() }
        active -> expired { expired() }
        inactive -> active { active() }
        expired -> active { active() }
    end
end

association HasBorrowed between
    Member[0..1] role borrower
    Copy[*] role borrowed
end

association HasReserved between
    Member[0..1] role reserver
    Copy[*] role reserved

```

```

end

association CopyOf between
    Copy[1..*] role copies
    Book[1] role book
end

constraints

context Member::borrow(c:Copy)
    pre cond1: status = #active
    pre limit: self.no_onloan < 2
    pre cond2: self.borrowed->excludes(c)
    post cond3: self.borrowed->includes(c)
    post status: c.status=#onLoan

context Member::reserve(c:Copy)
    pre cond1: status = #active
    pre limit: self.no_reserved < 2
    pre cond2: self.reserved->excludes(c)
    post cond3: self.reserved->includes(c)
    post status: c.status=#onReserve

context Member::unreserve(c:Copy)
    pre cond1: status = #active
    pre cond2: self.reserved->includes(c)
    post cond3: self.reserved->excludes(c)
    post status: c.status=#onShelf

context Member::return(c:Copy)
    pre cond1: status = #active
    pre cond2: self.borrowed->includes(c)
    post cond3: self.borrowed->excludes(c)
    post status: c.status=#onShelf

context Member::expired()
    pre status: status <> #expired

context Book::setRating(r:Integer)
    pre cond1: r >= 0 and r <= 10
    post cond2: self.rating = r

context Member::setBookRating(b:Book, r:Integer)
    pre cond1: status = #active
    pre cond2: r >= 0 and r <= 10
    post cond3: b.rating = r

```


The Soil Statements:

```
-- Script generated by USE 4.1.1

!new Book('b1')
!b1.title := 'The Hobbit'
!b1.author := 'J.R.R. Tolkien'

!new Book('b2')
!b2.title := 'The Hunger Games'
!b2.author := 'Suzanne Collins'

!new Book('b3')
!b3.title := 'Catching Fire'
!b3.author := 'Suzanne Collins'

!new Member('Ian')
!Ian.name := 'Ian'
!Ian.address := '274 North Circular'

!new Member('Lebron')
!Lebron.name := 'Lebron James'
!Lebron.address := 'Miami'

!new Member('Michael')
!Michael.name := 'Michael Jordan'
!Michael.address := 'Chicago'

!new Copy('c1')
!c1.book := b1
!insert(c1, b1) into CopyOf

!new Copy('c2')
!c2.book := b2
!insert(c2, b2) into CopyOf

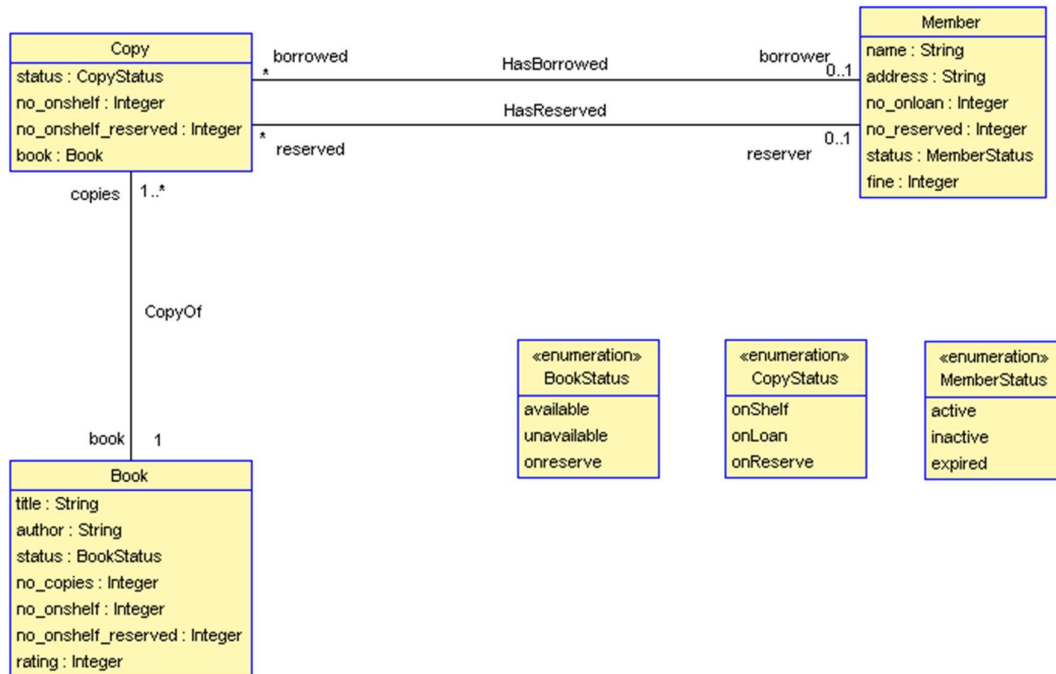
!new Copy('c3')
!c3.book := b3
!insert(c3, b3) into CopyOf
```

Opening Soil File:

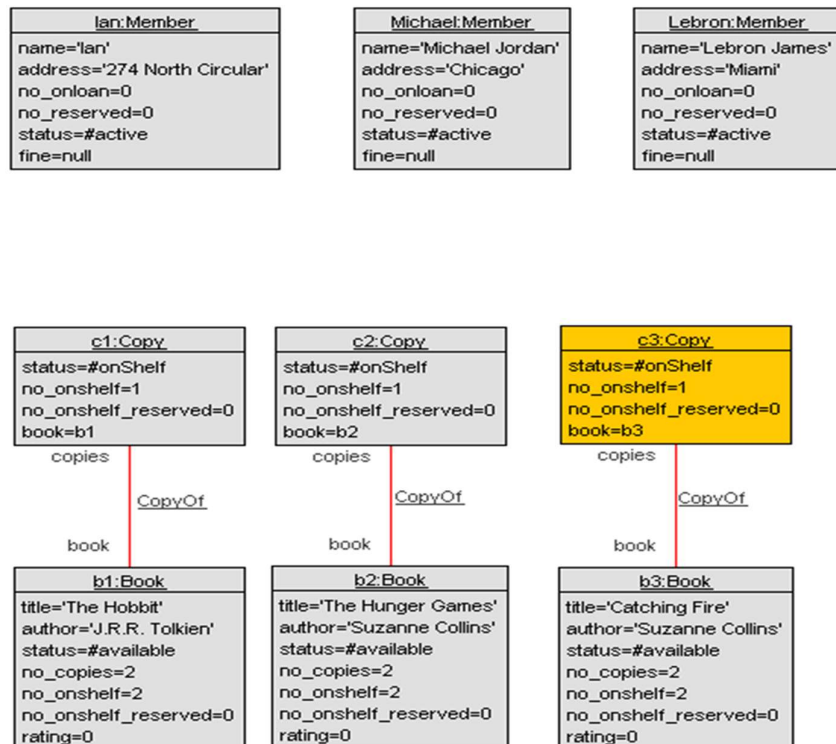
```
USE version 6.0.0, Copyright (C) 1999-2021 University
use> open LibAssign.soil
LibAssign.soil> -- Script generated by USE 4.1.1
LibAssign.soil>
LibAssign.soil> !new Book('b1')
LibAssign.soil> !b1.title := 'The Hobbit'
LibAssign.soil> !b1.author := 'J.R.R. Tolkien'
LibAssign.soil>
LibAssign.soil> !new Book('b2')
LibAssign.soil> !b2.title := 'The Hunger Games'
LibAssign.soil> !b2.author := 'Suzanne Collins'
LibAssign.soil>
LibAssign.soil> !new Book('b3')
LibAssign.soil> !b3.title := 'Catching Fire'
LibAssign.soil> !b3.author := 'Suzanne Collins'
LibAssign.soil>
LibAssign.soil> !new Member('Ian')
LibAssign.soil> !Ian.name := 'Ian'
LibAssign.soil> !Ian.address := '274 North Circular'
LibAssign.soil>
LibAssign.soil> !new Member('Lebron')
LibAssign.soil> !Lebron.name := 'Lebron James'
LibAssign.soil> !Lebron.address := 'Miami'
LibAssign.soil>
LibAssign.soil> !new Member('Michael')
LibAssign.soil> !Michael.name := 'Michael Jordan'
LibAssign.soil> !Michael.address := 'Chicago'
LibAssign.soil>
LibAssign.soil> !new Copy('c1')
LibAssign.soil> !c1.book := b1
LibAssign.soil> !insert(c1, b1) into CopyOf
LibAssign.soil>
LibAssign.soil> !new Copy('c2')
LibAssign.soil> !c2.book := b2
LibAssign.soil> !insert(c2, b2) into CopyOf
LibAssign.soil>
LibAssign.soil> !new Copy('c3')
LibAssign.soil> !c3.book := b3
LibAssign.soil> !insert(c3, b3) into CopyOf
LibAssign.soil>
LibAssign.soil>
LibAssign.soil>
LibAssign.soil>
use>
```

Screen Captures of The Diagrams of The Use Code:

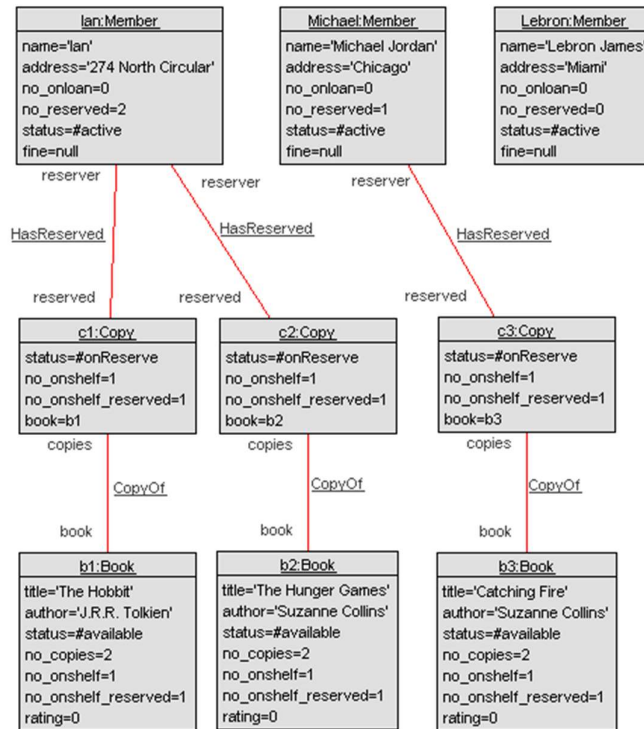
Class Diagram:



Object Diagram Before Any Testing:



Object Diagram (Reserve):



Soil Implementation Testing Reserve Operation:

```

use> !Ian.reserve(c1)
use> !Ian.reserve(c2)
use> !Michael.reserve(c3)
use> !Ian.reserve(c1)#
<input>:line 1:15 extraneous input '#' expecting EOF
use> !Ian.reserve(c1)
[Error] 2 preconditions in operation call `Member::reserve(self:Ian, c:c1)` do not hold:
  limit: (self.no_reserved < 2)
    self : Member = Ian
    self.no_reserved : Integer = 2
    2 : Integer = 2
    (self.no_reserved < 2) : Boolean = false

  cond2: self.reserved->excludes(c)
    self : Member = Ian
    self.reserved : Set(Copy) = Set{c1,c2}
    c : Copy = c1
    self.reserved->excludes(c) : Boolean = false

call stack at the time of evaluation:
  1. Member::reserve(self:Ian, c:c1) [caller: Ian.reserve(c1)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with '?', ':', 'help' or 'info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::reserve(self:Ian, c:c1)`.
use> !Ian.reserve(c3)
[Error] 1 precondition in operation call `Member::reserve(self:Ian, c:c3)` does not hold:
  limit: (self.no_reserved < 2)
    self : Member = Ian
    self.no_reserved : Integer = 2
    2 : Integer = 2
    (self.no_reserved < 2) : Boolean = false

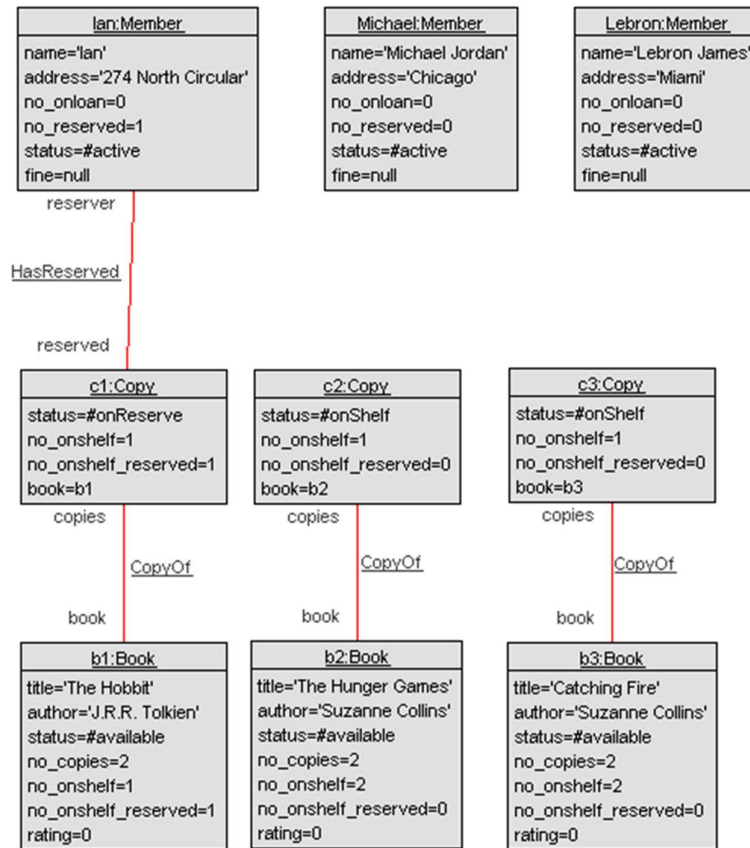
call stack at the time of evaluation:
  1. Member::reserve(self:Ian, c:c3) [caller: Ian.reserve(c3)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with '?', ':', 'help' or 'info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::reserve(self:Ian, c:c3)`.
  
```

Object Diagram (Unreserve):



Soil Implementation Testing Unreserve Operation:

```

use> !Ian.unreserve(c2)
use> !Michael.unreserve(c3)
use> !Michael.unreserve(c3)
[Error] 1 precondition in operation call 'Member::unreserve(self:Michael, c:c3)' does not hold:
  cond2: self.reserved->includes(c)
    self : Member = Michael
    self.reserved : Set(Copy) = Set{}
    c : Copy = c3
    self.reserved->includes(c) : Boolean = false

call stack at the time of evaluation:
  1. Member::unreserve(self:Michael, c:c3) [caller: Michael.unreserve(c3)<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with '?', ':', 'help' or 'info' are allowed.
'c' continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call 'Member::unreserve(self:Michael, c:c3)'.
use> !Michael.unreserve(c1)
[Error] 1 precondition in operation call 'Member::unreserve(self:Michael, c:c1)' does not hold:
  cond2: self.reserved->includes(c)
    self : Member = Michael
    self.reserved : Set(Copy) = Set{}
    c : Copy = c1
    self.reserved->includes(c) : Boolean = false

call stack at the time of evaluation:
  1. Member::unreserve(self:Michael, c:c1) [caller: Michael.unreserve(c1)<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

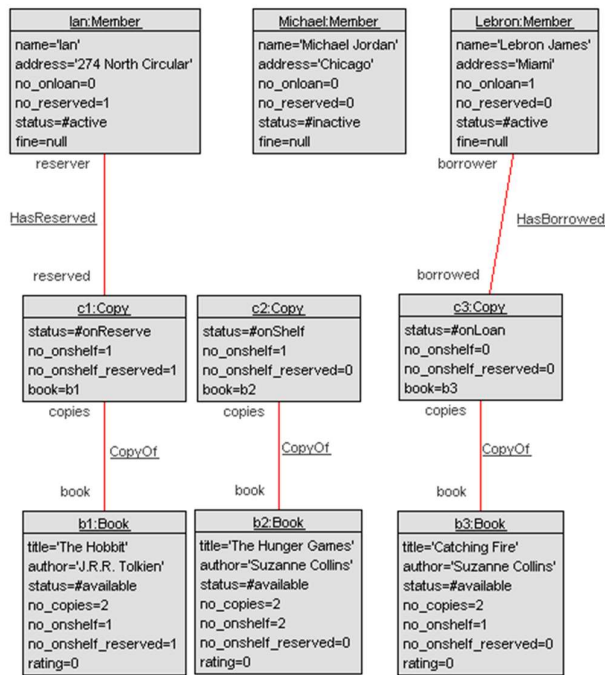
Currently only commands starting with '?', ':', 'help' or 'info' are allowed.
'c' continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call 'Member::unreserve(self:Michael, c:c1)'.
use>

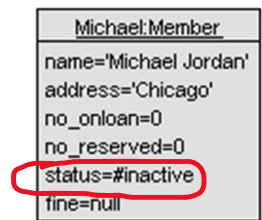
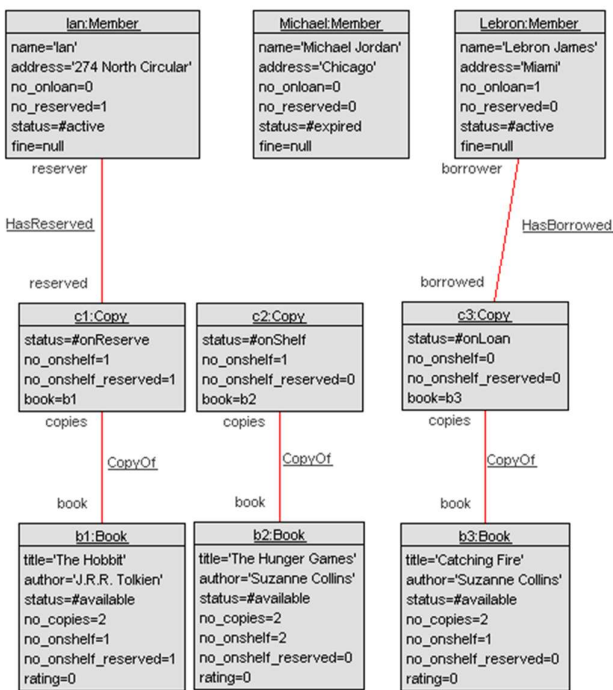
```


Object Diagram (Inactive and Expired):

Inactive Status (Michael):



Expired Status (Michael):



Testing Inactive and Expired Operation:

Soil Implementation Inactive():

```
use> !Michael.inactive()
use> !Michael.reserve(c2)
[Error] 1 precondition in operation call `Member::reserve(self:Michael, c:c2)` does not hold:
  cond1: (self.status = MemberStatus::active)
    self : Member = Michael
    self.status : MemberStatus = MemberStatus::inactive
    MemberStatus::active : MemberStatus = MemberStatus::active
    (self.status = MemberStatus::active) : Boolean = false

  call stack at the time of evaluation:
    1. Member::reserve(self:Michael, c:c2) [caller: Michael.reserve(c2)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::reserve(self:Michael, c:c2)`.
use> !Michael.borrow(c2)
[Error] 1 precondition in operation call `Member::borrow(self:Michael, c:c2)` does not hold:
  cond1: (self.status = MemberStatus::active)
    self : Member = Michael
    self.status : MemberStatus = MemberStatus::inactive
    MemberStatus::active : MemberStatus = MemberStatus::active
    (self.status = MemberStatus::active) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:Michael, c:c2) [caller: Michael.borrow(c2)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::borrow(self:Michael, c:c2)`.
use>
```

Soil Implementation Expired():

```
use> !Michael.expired()
use> !Michael.reserve(c2)
[Error] 1 precondition in operation call `Member::reserve(self:Michael, c:c2)` does not hold:
  cond1: (self.status = MemberStatus::active)
    self : Member = Michael
    self.status : MemberStatus = MemberStatus::expired
    MemberStatus::active : MemberStatus = MemberStatus::active
    (self.status = MemberStatus::active) : Boolean = false

  call stack at the time of evaluation:
    1. Member::reserve(self:Michael, c:c2) [caller: Michael.reserve(c2)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::reserve(self:Michael, c:c2)`.
use> !Michael.borrow(c2)
[Error] 1 precondition in operation call `Member::borrow(self:Michael, c:c2)` does not hold:
  cond1: (self.status = MemberStatus::active)
    self : Member = Michael
    self.status : MemberStatus = MemberStatus::expired
    MemberStatus::active : MemberStatus = MemberStatus::active
    (self.status = MemberStatus::active) : Boolean = false

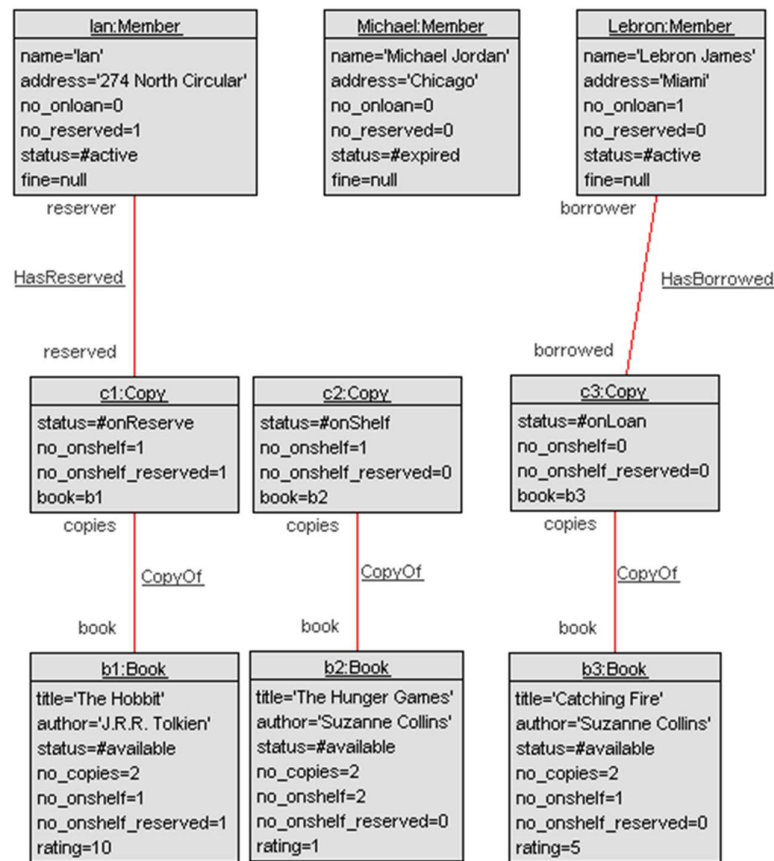
  call stack at the time of evaluation:
    1. Member::borrow(self:Michael, c:c2) [caller: Michael.borrow(c2)@<input>:1:0]

+-----+
| Evaluation is paused. You may inspect, but not modify the state. |
+-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::borrow(self:Michael, c:c2)`.
use>
```

Object Diagram (setRating and setBookRating):



Soil Implementation Testing setRating and setBookRating:

```
libAssign:Soil/
use> !Ian.borrow(c1)
use> !Ian.setBookRating(b1, 10)
use> !Ian.setBookRating(b1, 20)
[Error] 1 precondition in operation call `Member::setBookRating(self:Ian, b:b1, r:20)` does not hold:
  cond2: ((r >= 0) and (r <= 10))
    r : Integer = 20
    0 : Integer = 0
    (r >= 0) : Boolean = true
    r : Integer = 20
    10 : Integer = 10
    (r <= 10) : Boolean = false
    ((r >= 0) and (r <= 10)) : Boolean = false

call stack at the time of evaluation:
  1. Member::setBookRating(self:Ian, b:b1, r:20) [caller: Ian.setBookRating(b1, 20)@<input>:1:0]

-----+
| Evaluation is paused. You may inspect, but not modify the state. |
-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::setBookRating(self:Ian, b:b1, r:20)`.
use> !Ian.setBookRating(b1, -10)
[Error] 1 precondition in operation call `Member::setBookRating(self:Ian, b:b1, r:-10)` does not hold:
  cond2: ((r >= 0) and (r <= 10))
    r : Integer = -10
    0 : Integer = 0
    (r >= 0) : Boolean = false
    ((r >= 0) and (r <= 10)) : Boolean = false

call stack at the time of evaluation:
  1. Member::setBookRating(self:Ian, b:b1, r:-10) [caller: Ian.setBookRating(b1, -10)@<input>:1:0]

-----+
| Evaluation is paused. You may inspect, but not modify the state. |
-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).

> c
Error: precondition false in operation call `Member::setBookRating(self:Ian, b:b1, r:-10)`.
use> !Ian.setBookRating(b2, 5)
[Error] 1 postcondition in operation call `Member::setBookRating(self:Ian, b:b2, r:5)` does not hold:
  cond3: (b.rating = r)
    b : Book = b2
    b.rating : Integer = 0
    r : Integer = 5
    (b.rating = r) : Boolean = false

call stack at the time of evaluation:
  1. Member::setBookRating(self:Ian, b:b2, r:5) [caller: Ian.setBookRating(b2, 5)@<input>:1:0]


-----+
| Evaluation is paused. You may inspect, but not modify the state. |
-----+

Currently only commands starting with `?`, `:`, `help` or `info` are allowed.
`c` continues the evaluation (i.e. unwinds the stack).
```

Soil Implementation Testing !Openter and !Opexit with Reserve Operation:


```
use> !openter Ian reserve(c1)
precondition `cond1' is true
precondition `limit' is true
precondition `cond2' is true
use> !insert(Ian, c1) into HasReserved
use> !c1.status := #onReserve
use> !Ian.status := #active
use> !opexit
postcondition `cond3' is true
postcondition `status' is true
use> _
```

OCL Contracts:

 Evaluate OCL expression ✕


Enter OCL expression:

Result:

 Evaluate OCL expression ✕


Enter OCL expression:

Result:

 Evaluate OCL expression ✕

Enter OCL expression:

Result:

 Evaluate OCL expression ✕

Enter OCL expression:

b3.rating

Result:


5 : Integer

Evaluate

Browser

Clear

Close

 Evaluate OCL expression ✕

Enter OCL expression:

Michael.status=#active

Result:

false : Boolean

Evaluate

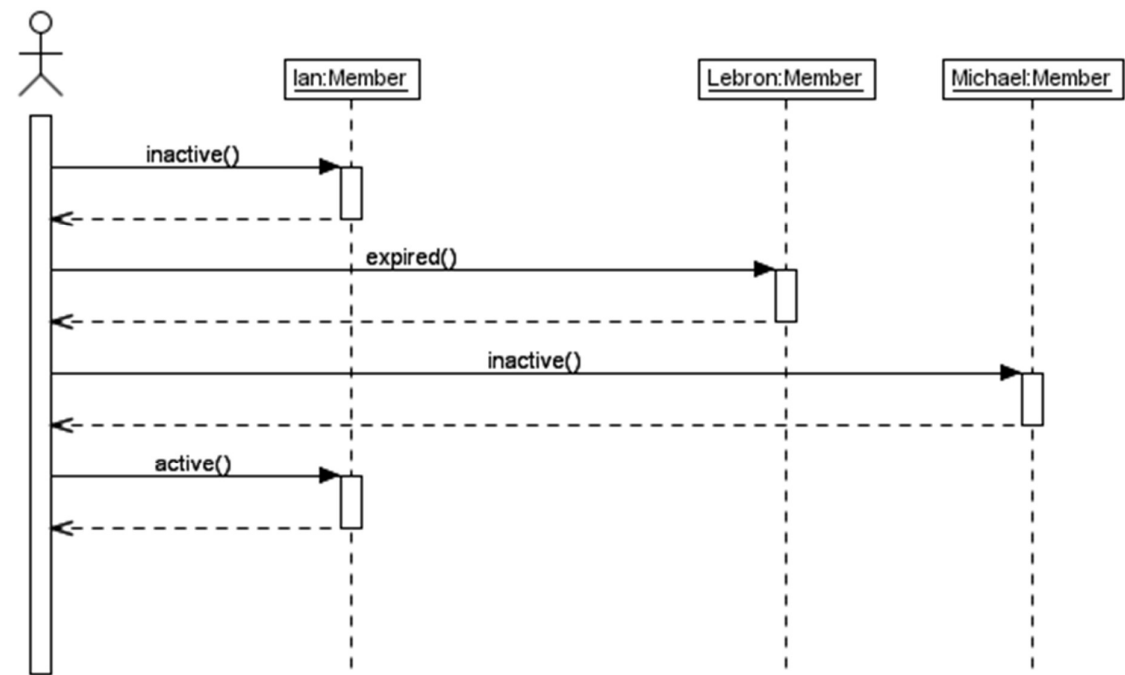
Browser

Clear

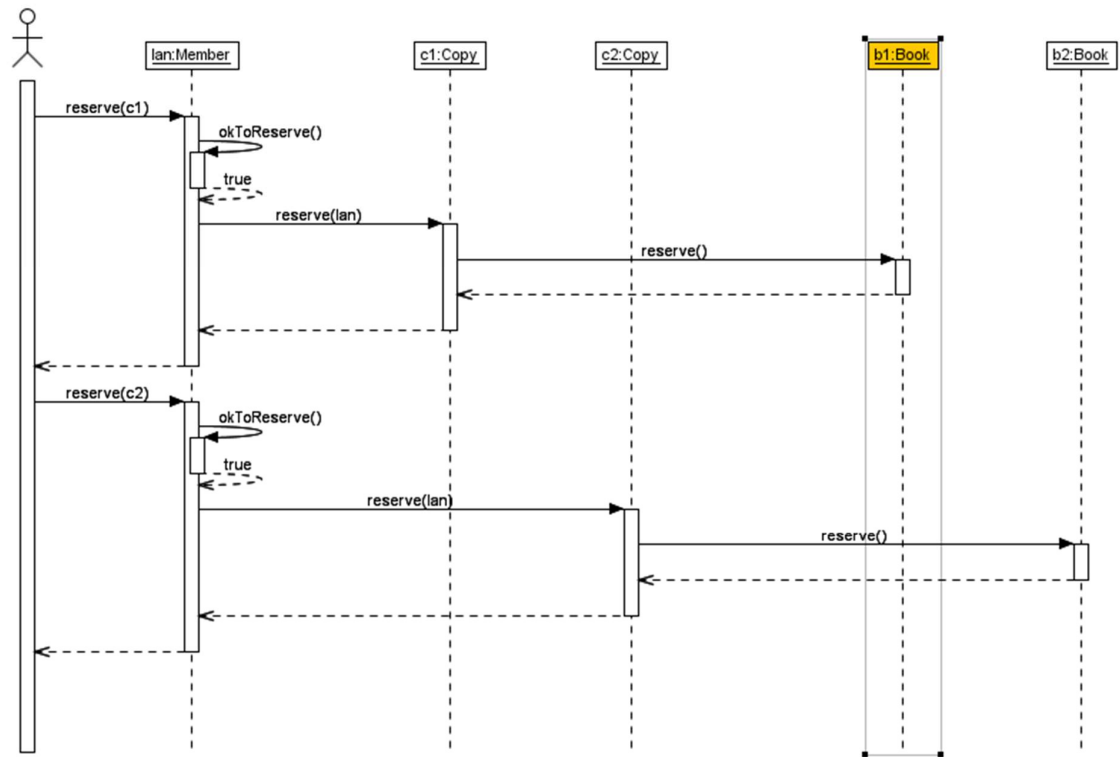
Close

Sequence Diagrams:

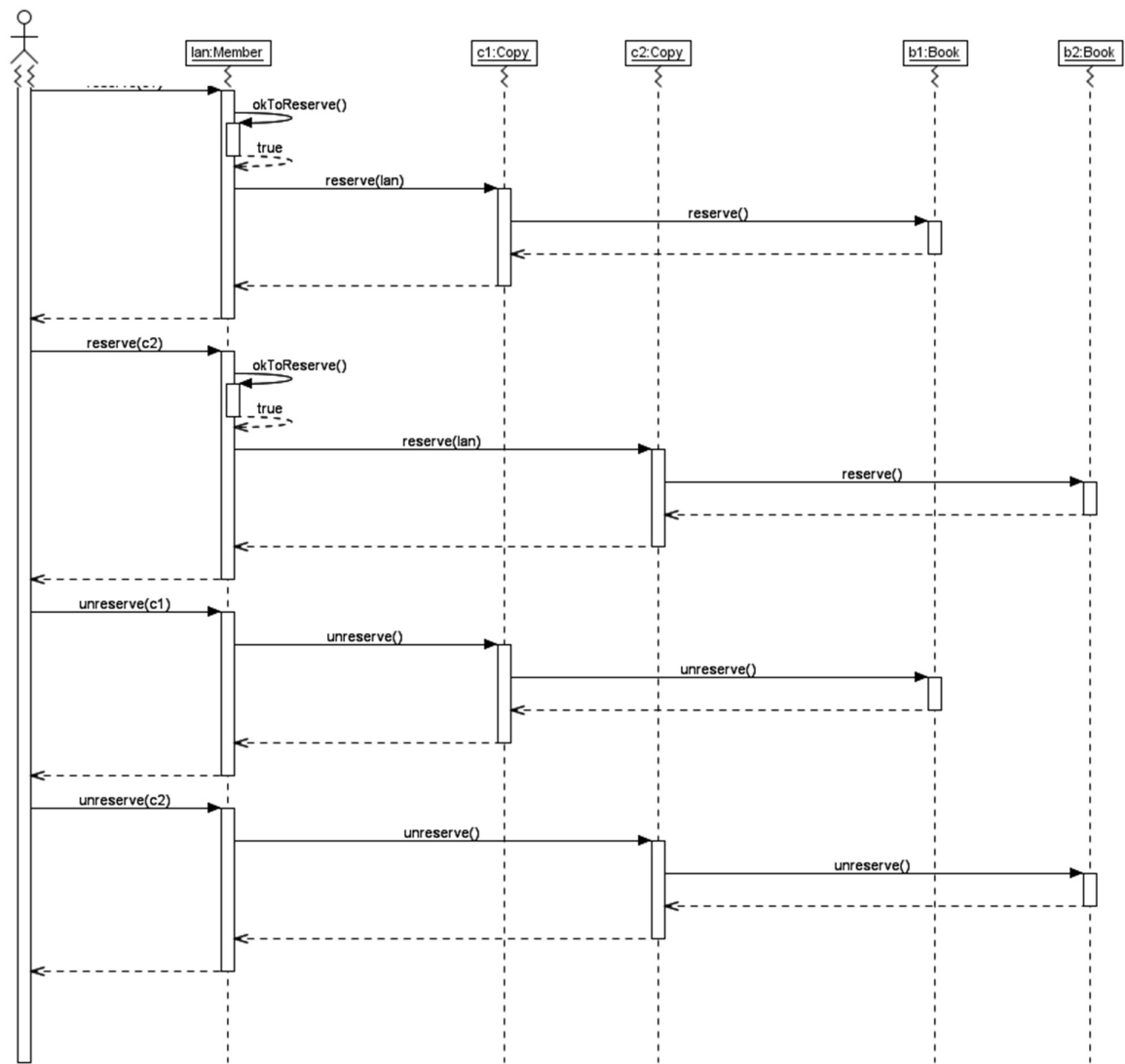
Member Status:



Reserve:

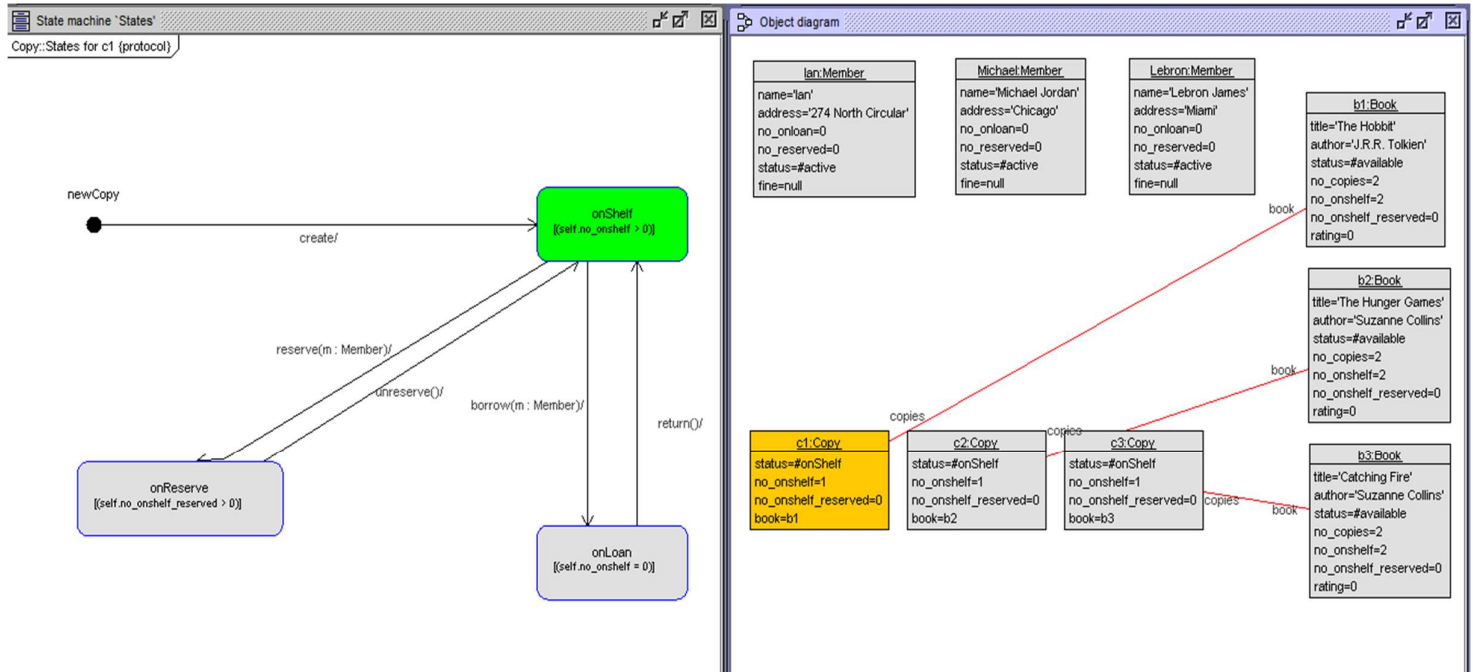


Unreserve:

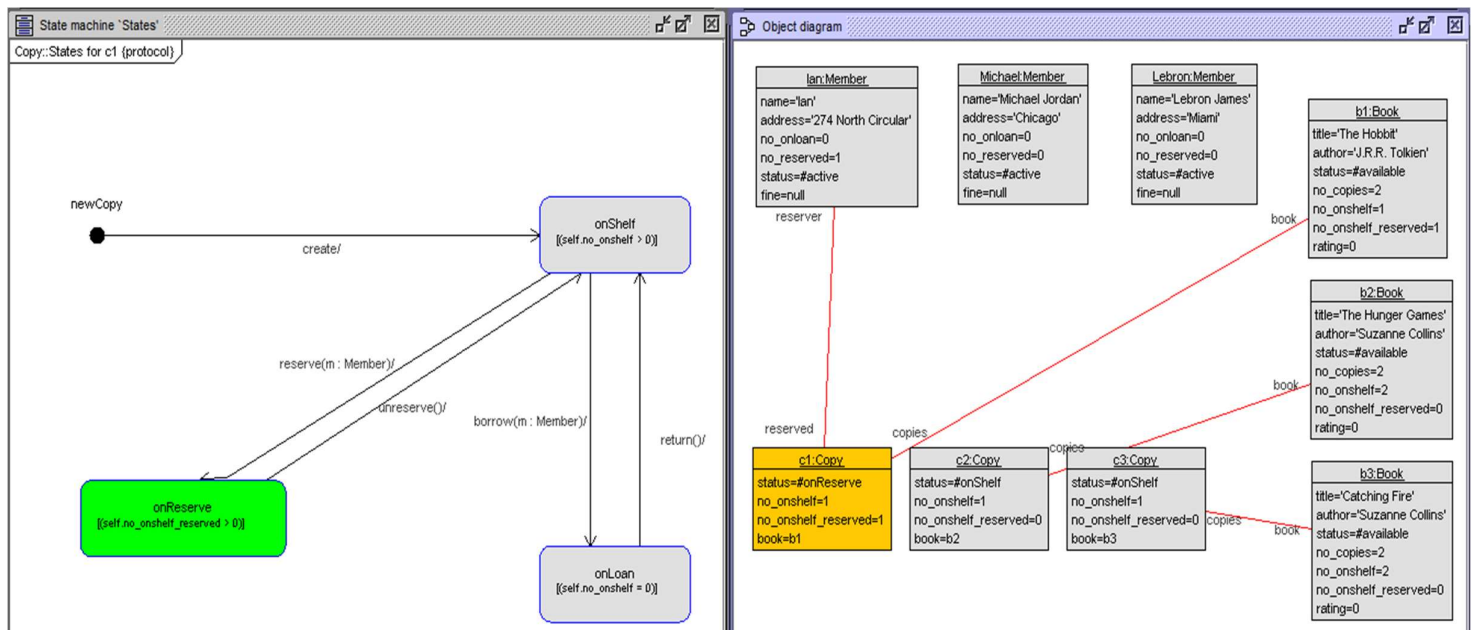


Statemachines:

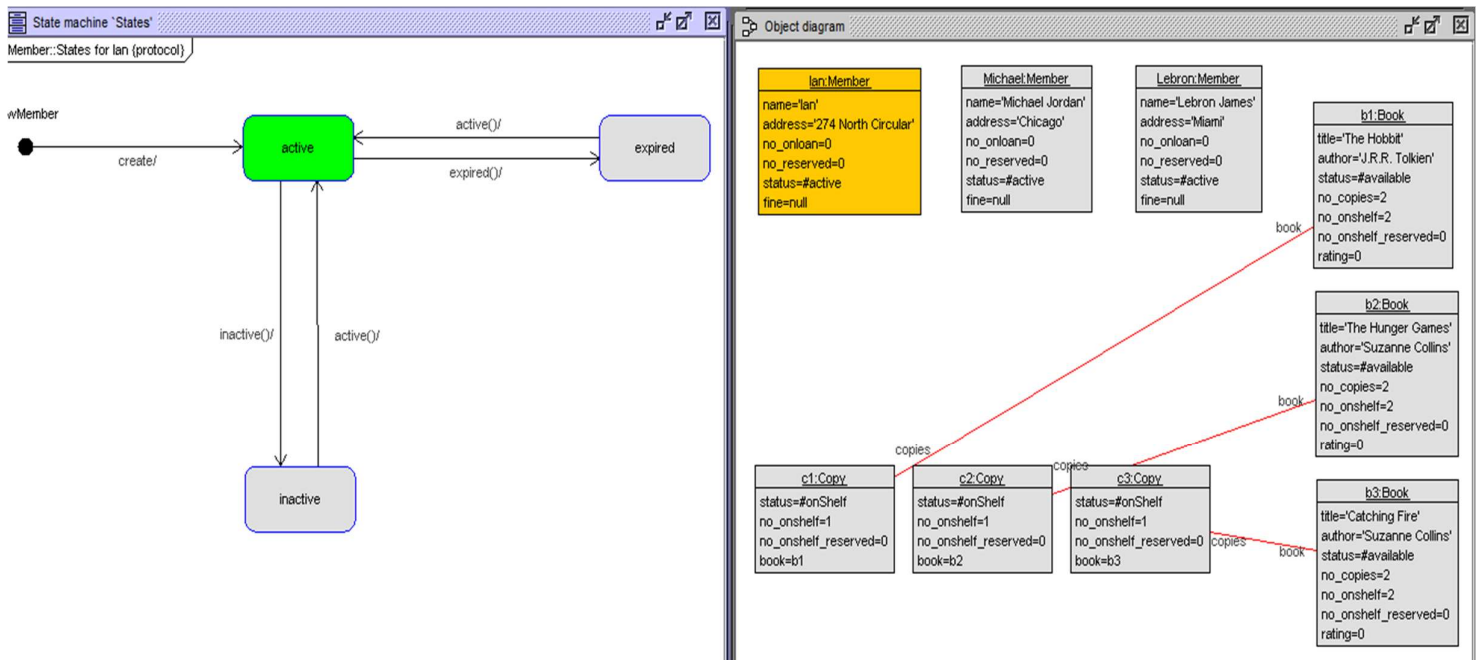
Before Reserve:



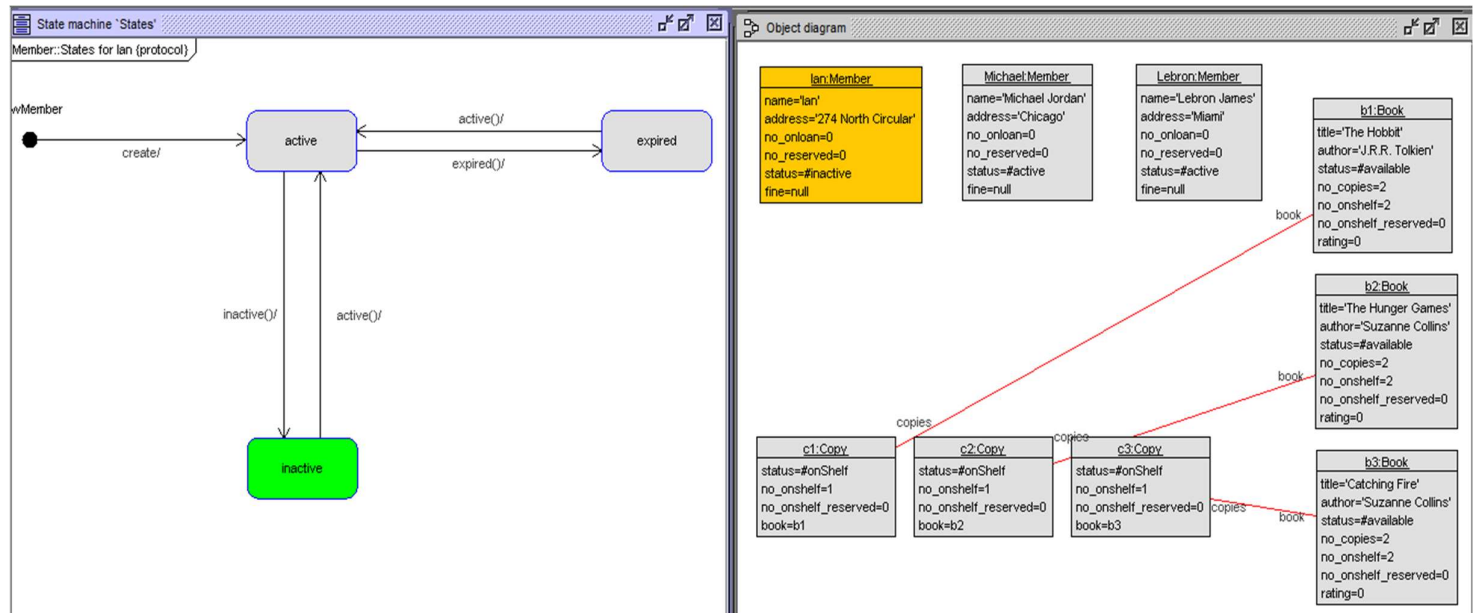
After Reserve:



Before (Active):



After (Inactive):



Discussion and Analysis:

This report discusses the extension and testing of a more comprehensive USE model for the library system. I have provided an explanation of all the new use cases (reserve, unreserve, member management system and rating system) that I have created and implemented. I have also included screen captures of different diagrams, class diagram, object diagrams, sequence diagrams, OCL contracts, and statemachine testing. Accompanied by these diagrams is the testing of the functionality of the use cases in SOIL that I have implemented. These screen captures give a good analysis of all the new use cases.

Creating the reserve operation expanded my understanding of the reservation process as I had to learn how to implement it in all three of the classes. It also gave me firsthand experience in designing and implementing reservation processes such as the intricacies involved in managing reservations, handling all the requirements, errors, etc. when creating this use case. I can say this for all the new use cases I implemented.

Overall, by completing this assignment I have gained a good understanding and knowledge of how to implement new use cases in a USE model. I learnt how to carefully consider the USE model to know what use cases to implement and what would work best in the library system. Implementing these new use cases and working on the code helped me to figure out how to work with USE and SOIL and gave me valuable knowledge on how to model and design a system while developing the code. I also gained skills in testing and identifying potential issues, as implementing the new use cases required comprehensive testing to ensure everything worked correctly and smoothly, while also ensuring the overall quality of the library system.