Library Use Model Report

Introduction

For this assignment, I collaborated with (student) to develop the USE model for the library. We decided to choose the assignment "Extend and test a more comprehensive USE model for the Library system in USE". The library USE file given had the classes Book, Copy and Member. In this report I will discuss how I implemented the Reservation class, created the USE cases for reserving, unreserving, pay fine and extend loan as well as the necessary state machines. The model was tested with SOIL scripts and validated through object diagrams, state diagrams, and sequence diagrams. At least one operation was tested with *!opexit* and *!openter*.

This assignment was quite a challenge to model as we had to model the meaning of what a reservation is particularly and how it would work with the different states for the copies and the borrowing system. I had to ensure that the logic, object associations and the constraints were consistent and had to think carefully about the state machines and the preconditions.

Use Cases Introduced

The following use cases were introduced in addition to standard borrow and return operations:

Reserve a Copy
This use case allows the member to reserve a copy of a book that is available on the shelf. It makes sure that the reserved copy is marked as reserved and cannot be borrowed by another member. The system makes sure that only one reservation can exist per copy at a time and that is only if it isn't already either reserved or borrowed. A new reservation object is created and it links to Member and Copy. To enforce this, we created a new Reservation class. Only books on the shelf can be reserved. The reservation status of both the Copy and Reservation object must be updated. The state machine tracks the reservation's lifecycle.

Remove a Reservation
This use case allows the member to cancel a reservation. The system will check if the member has a reservation on the chosen copy and if so the cancel operation will work. This will reset the copy's reservation status and remove the links it has to the objects. Only the member that made the reservation can remove it. The postcondition ensures that the reservation is removed and the copy can be available again. A proper error messaged

Pay a Fine
This use case allows the member to pay a fine in either full or a part of it. The system checks the amount and updates the member's balance. !*openter* and !*opexit* are used for this part. This validates a fine only if the fine is positive integer. It allows partial payments without going below 0 and provides error messages if a payment is invalid.

Renew a Loan
Member can renew the loan on a borrowed copy if it is still in their possession. Preconditions check that the copy is borrowed by the member and is currently on loan.

USE Code

```
model Library

enum BookStatus { available, unavailable, onreserve }
enum CopyStatus { onLoan, onShelf, onReserve }
enum ReserveStatus { Reserved, NotReserved }

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

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

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

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

class Copy
    attributes
        status : CopyStatus init = #onShelf
        reserved : ReserveStatus init = #NotReserved

    operations
        return()
        begin
            self.status := #onShelf;
            self.book.return()
        end

        reserve(m: Member)
        begin
```

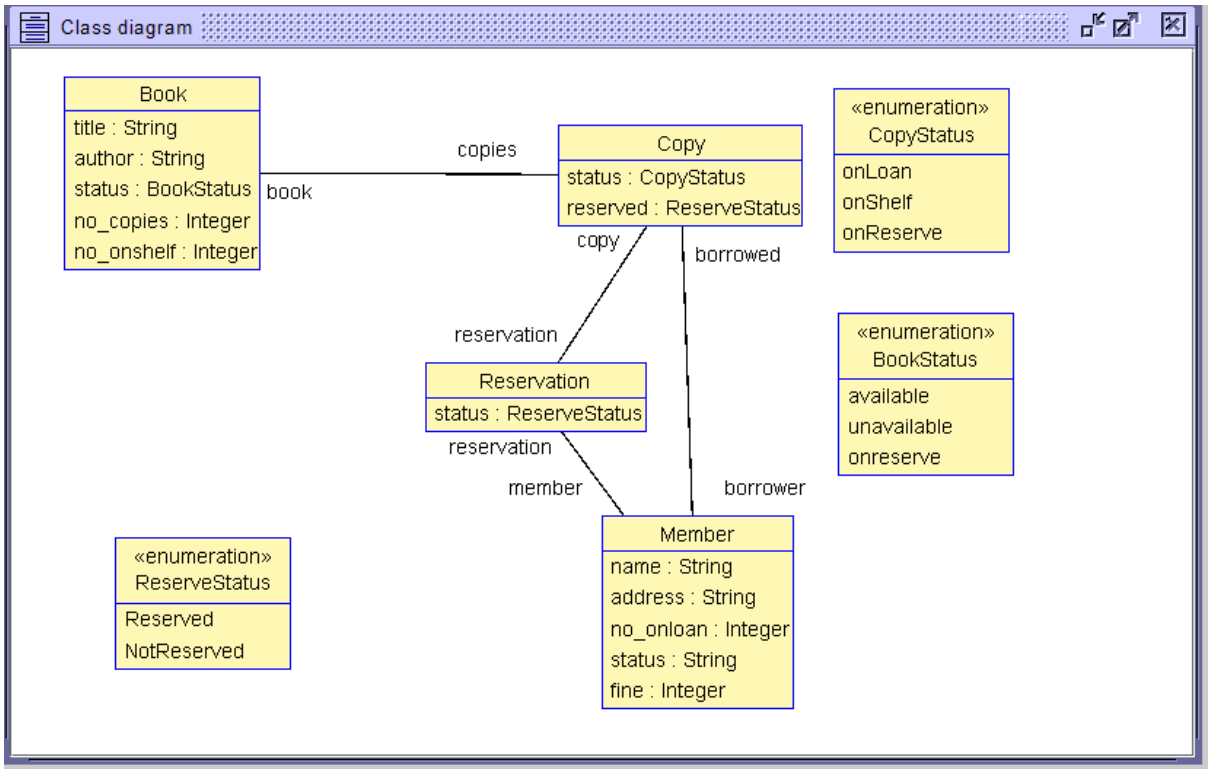
```
            self.reserved := #Reserved;
            WriteLine('This copy has been reserved. Please manually
create and link a Reservation object.');
end



removeReservation(m: Member)
begin
    if self.reserved = #NotReserved then
        WriteLine('This Copy does not have a reservation to remove');
    else
        self.reserved := #NotReserved;

        for r in Reservation.allInstances do
            if r.copy = self and r.status = #Reserved then
                r.cancel();
            end
        end;

        WriteLine('This book can now be reserved');
    end
end




        borrow(m: Member)
        begin
            self.status := #onLoan;
            self.book.borrow()
        end

        cancelReservation()
        begin
            for r in Reservation.allInstances do
                if r.copy = self and r.status = #Reserved then
                    r.cancel();
                end
            end;
        end


    statemachines
        psm States
        states
            newCopy : initial
            onLoan
            onShelf
            onReserve
        transitions
            newCopy -> onShelf  { create }
            onShelf -> onLoan { borrow() }
            onLoan -> onShelf { return() }
            onShelf -> onReserve { reserve() }
            onReserve -> onLoan { borrow() }
            onReserve -> onShelf   { cancelReservation() }

        end
end
```

```
class Member
    attributes
        name : String
        address : String
        no_onloan : Integer
        status : String
        fine : Integer
    operations
        okToBorrow() : Boolean
        begin
            if (self.no_onloan < 2) 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

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


        viewBorrowed()
        begin
            for c in self.borrowed do
                WriteLine(c.book.title);
            end;
        end

        reserve(c: Copy)
        begin
            c.reserve(self)
        end

        removeReservation(c: Copy)
        begin
            c.removeReservation(self)
        end

        payFine(amount: Integer)
        begin
            if amount > 0 and self.fine > 0 then
                if amount >= self.fine then
                    self.fine := 0
                else
                    self.fine := self.fine - amount
```

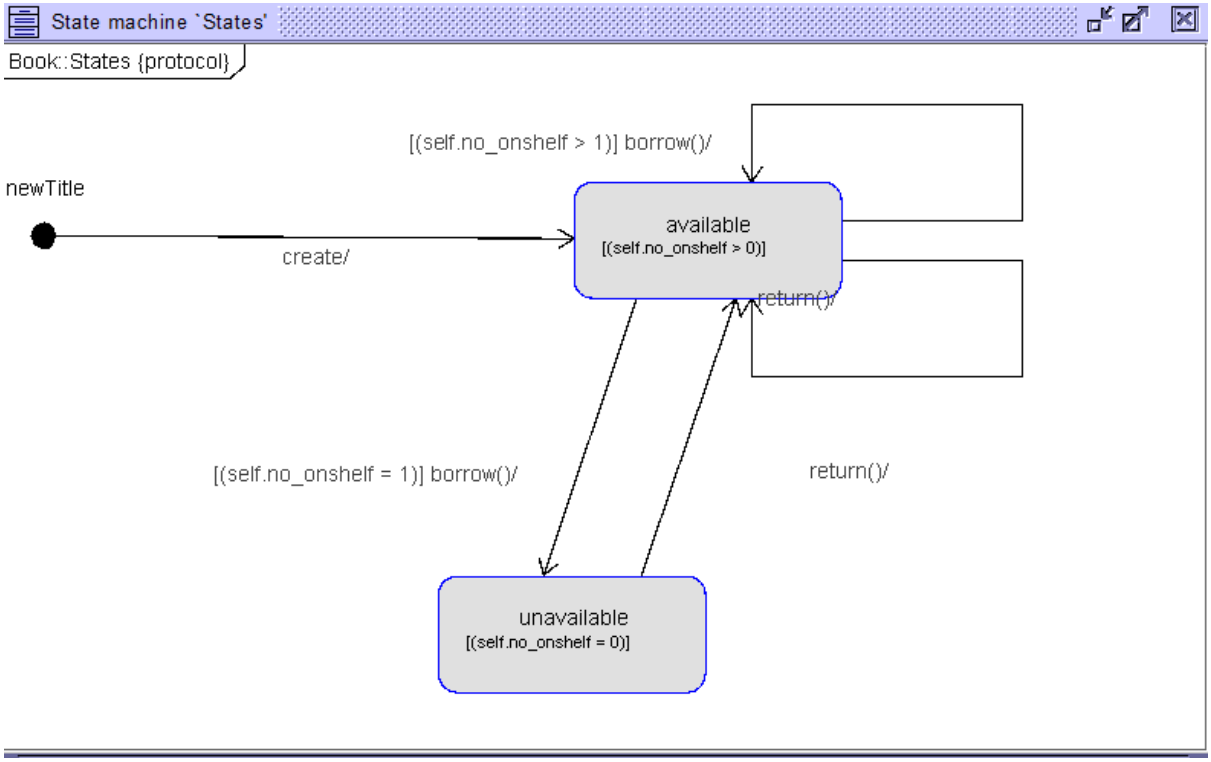
```
                end
            else
                WriteLine('Cannot process payment: no fine or invalid
amount');
        end
        end

        renewLoan(c: Copy)
            pre hasCopy: self.borrowed->includes(c)
            pre copyIsOnLoan: c.status = #onLoan
            post stillHasCopy: self.borrowed->includes(c)

end

class Reservation
    attributes
        status : ReserveStatus init = #NotReserved

    operations
    cancel()
    begin
        self.status := #NotReserved;
        self.copy.reserved := #NotReserved;

        if self.copy.status = #onReserve then
            self.copy.status := #onShelf;
        end;

        WriteLine('Reservation cancelled');
        delete (self, self.copy) from ForCopy;
        delete (self, self.member) from ReservedBy;
    end




    statemachines
        psm ReserveLifecycle
        states
            NotReserved : initial
            Reserved
            Cancelled
        transitions
            NotReserved -> Reserved { create }
            Reserved -> Cancelled { [status = #Reserved] cancel() }
        end
end



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

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

```
association ForCopy between
    Reservation[1] role reservation
    Copy[1] role copy
end


association ReservedBy between
    Reservation[1] role reservation
    Member[1] role member
end


constraints

context Member::borrow(c: Copy)
    pre limit: self.no_onloan < 2
    pre cond1: self.borrowed->excludes(c)
    post cond2: self.borrowed->includes(c)

context Copy
    inv statusInv: self.status = #onShelf or self.status = #onLoan or
self.status = #onReserve

context Member
    inv forbidden: self.borrowed.book.title->excludes('Staff Logs')

context Member::reserve(c: Copy)
pre noReservation: Reservation.allInstances->select(r | r.copy = c)-
>isEmpty()

context Copy::reserve(m: Member)
pre notReserved: self.reserved = #NotReserved
pre notAlreadyBorrowed: self.status = #onShelf

context Member::removeReservation(c: Copy)
pre hasReservation: Reservation.allInstances->exists(r | r.copy = c and
r.member = self)
post noReservation: Reservation.allInstances->select(r | r.copy = c and
r.member = self)->isEmpty()
```

Class Diagram



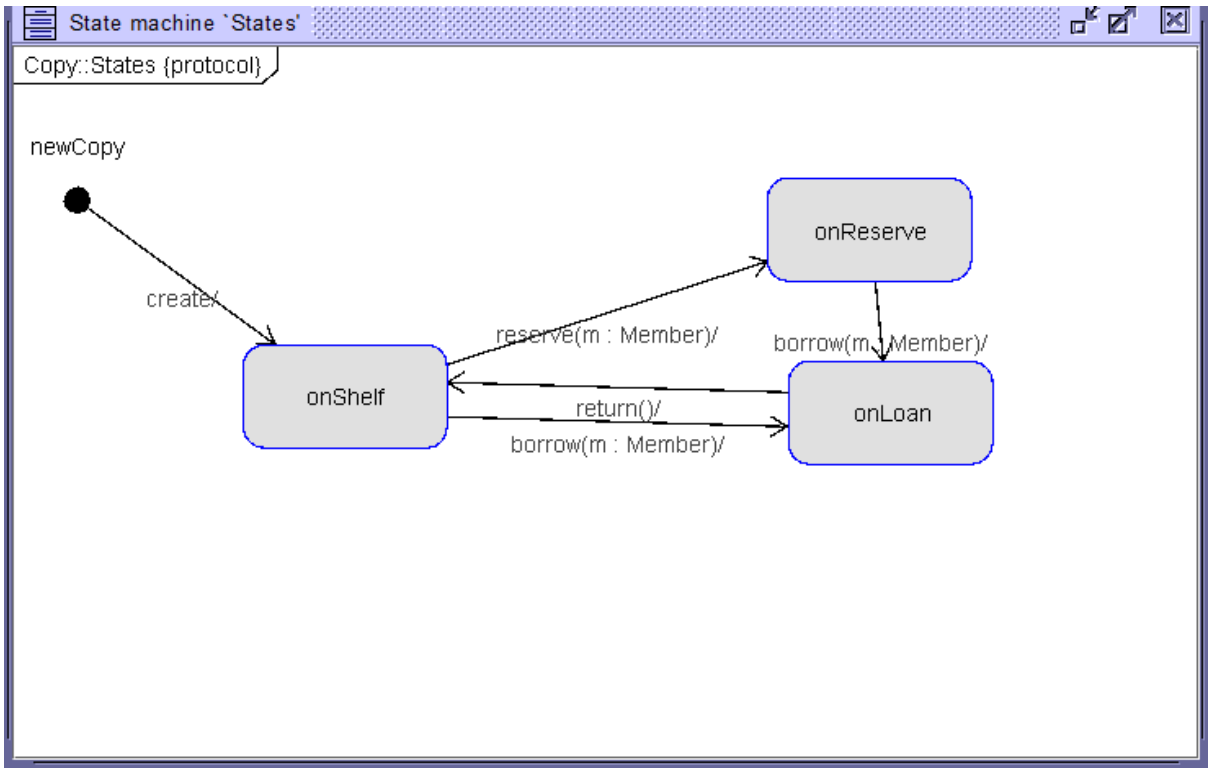
State Machines



USE Code for State Machine – Book

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

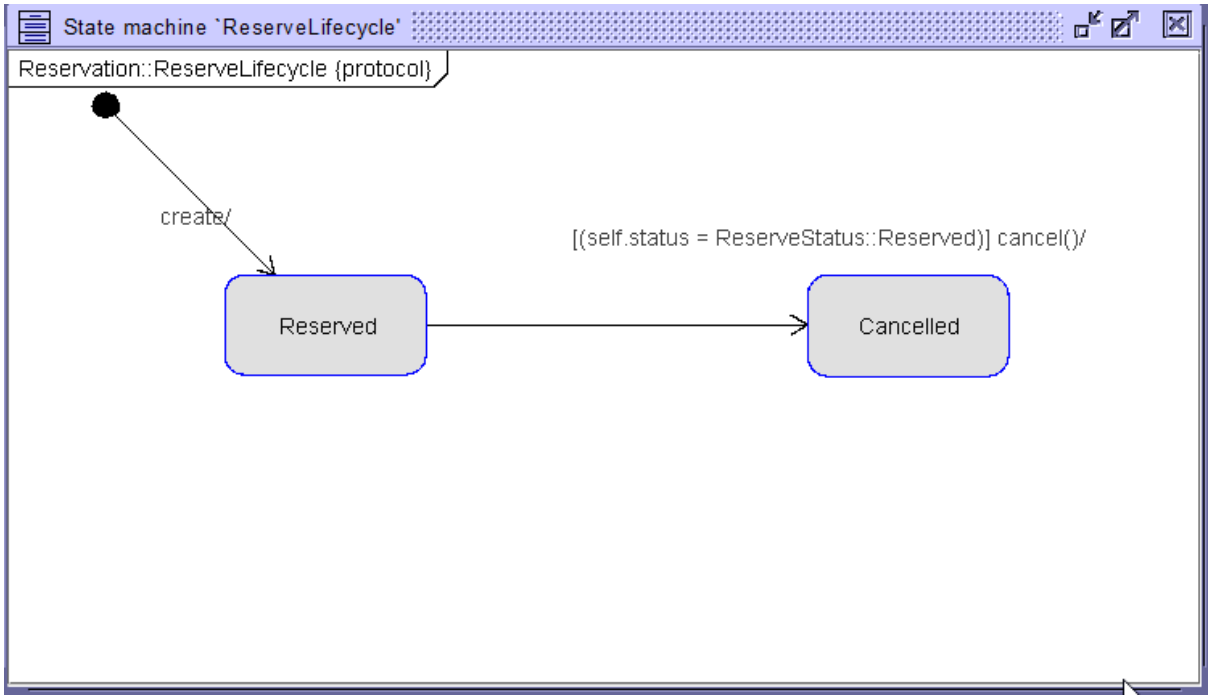
The state machine for the Book class models the transitions between the states NewTitle, available and unavailable based on how many copies are on the shelf. The book starts as NewTitle and then transitions to available when created. When a copy is borrowed, the book is unavailable if the number of copies is 0. If the book is returned it is available as the number of copies on the shelf is more than 0. The state machine makes sure the correct tracking of the book's status based on its availability for borrowing or returning.



USE Code for State Machine – Copy

```
    statemachines
        psm States
        states
            newCopy : initial
            onLoan
            onShelf
            onReserve
        transitions
            newCopy -> onShelf  { create }
            onShelf -> onLoan { borrow() }
            onLoan -> onShelf { return() }
            onShelf -> onReserve { reserve() }
            onReserve -> onLoan { borrow() }
            onReserve -> onShelf   { cancelReservation() }

        end
end
```

This state machine for the Copy class shows the transition between the states of newCopy, onShelf, onLoan and onReserve. It starts in the newCopy state and moves to onShelf when created. From there a copy can be borrowed moving it to onLoan or reserved and then transitioning to onReserve. The return() operation brings the copy back to onShelf and a reservation can be cancelled also returning it to onShelf. Each transition is controlled by borrow(), reserve() and cancelReservation().



USE Code for State Machine - Reservation

```
    statemachines
        psm ReserveLifecycle
        states
            NotReserved : initial
            Reserved
            Cancelled
        transitions
            NotReserved -> Reserved { create }
            Reserved -> Cancelled { [status = #Reserved]
cancel() }
        end
end
```

The state machine for the Reservation class shows the lifecycle of a reservation. It begins in the NotReserved state and transitions to Reserved when created. If the reservation is cancelled, it moves to the Cancelled state but only if the status of it is Reserved. The state machine makes sure that the reservations are properly managed and that transitions and conditions like cancel() are being triggered to cancel a reservation.

Constraints

The following constraints were implemented:

Borrowing Constraints

```
context Member::borrow(c: Copy)
pre limit: self.no_onloan < 2
pre cond1: self.borrowed->excludes(c)
post cond2: self.borrowed->includes(c)
```

- Limit: A member can borrow only if they have less than two books.
- cond1: A member cannot borrow a copy they already have.
- cond2: The borrowed copy is included with the other borrowed copies.

Copy Status

```
context Copy
inv statusInv: self.status = #onShelf or self.status =
#onLoan or self.status = #onReserve
```

- The copy must have a valid status: onShelf, onLoan, or onReserve.

Restrictions on types of books that can be borrowed

```
context Member
inv forbidden: self.borrowed.book.title->excludes('Staff
Logs')
```

- Members are not allowed to borrow the Staff Logs

Reservation Preconditions

```
context Member::reserve(c: Copy)
pre noReservation: c.reservation -> isEmpty()
```

- A copy can only be reserved if it hasn't already been reserved.

Reserving Copies

```
context Copy::reserve(m: Member)
pre notReserved: self.reserved = #NotReserved
pre notAlreadyBorrowed: self.status = #onShelf
```

- Only unreserved copies on the shelf can be reserved.

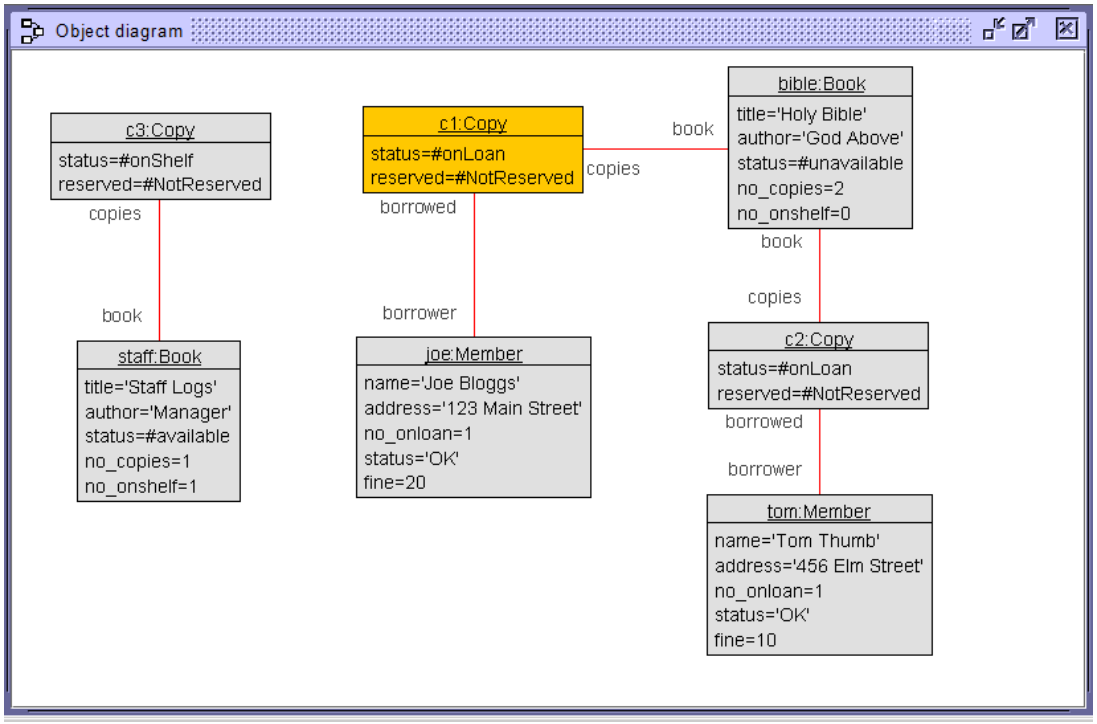
Removing a Reservation

```
context Member::removeReservation(c: Copy)
pre hasReservation: c.reservation -> includes(self)
post noReservation: c.reservation -> isEmpty()
```

- Members can only cancel their own reservations.
- After removing the reservations, the copy is unreserved.

SOIL Testing

Object Diagram



1. Forbidden to borrow Staff Logs

```
use> use> use>
use> !tom.borrow(c3)
[Error] 1 precondition in operation call 'Member::borrow(self:tom, c:c3)' does not hold:
  limit: (self.no_onloan < 2)
    self : Member = tom
    self.no_onloan : Integer = 2
    2 : Integer = 2
    (self.no_onloan < 2) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:tom, c:c3) [caller: tom.borrow(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).

> |
```

2. Cannot reserve borrowed books

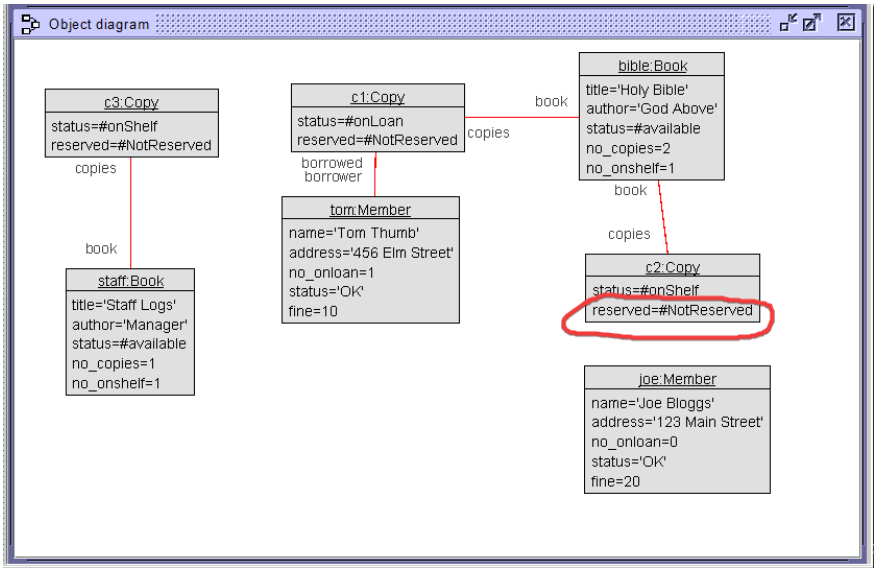
```
use> !tom.borrow(c1)
use> !joe.reserve(c1)
[Error] 1 precondition in operation call 'Copy::reserve(self:c1, m:joe)' does not hold:
  notAlreadyBorrowed: (self.status = CopyStatus::onShelf)
    self : Copy = c1
    self.status : CopyStatus = CopyStatus::onLoan
    CopyStatus::onShelf : CopyStatus = CopyStatus::onShelf
    (self.status = CopyStatus::onShelf) : Boolean = false
```

3. Can reserve unborrowed books

```
use> !joe.reserve(c2)
This copy has been reserved. Please manually create and link a Reservation object.
use> !new Reservation('r1')
use> !insert (r1, c1) into ForCopy
use> !insert (r1, joe) into ReservedBy
use> !r1.status := #Reserved
```

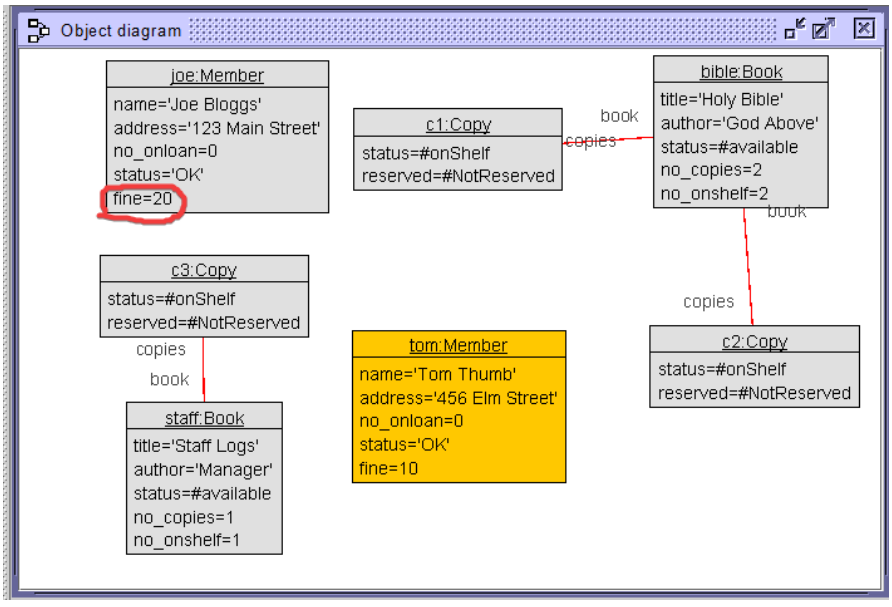
4. Cancel reservation

```
use> !joe.reserve(c2)
This copy has been reserved. Please manually create and link a Reservation object.
use> !new Reservation('r1')
use> !insert (r1, c1) into ForCopy
use> !insert (r1, joe) into ReservedBy
use> !r1.status := #Reserved
use> !r1.cancel()
Reservation cancelled
```



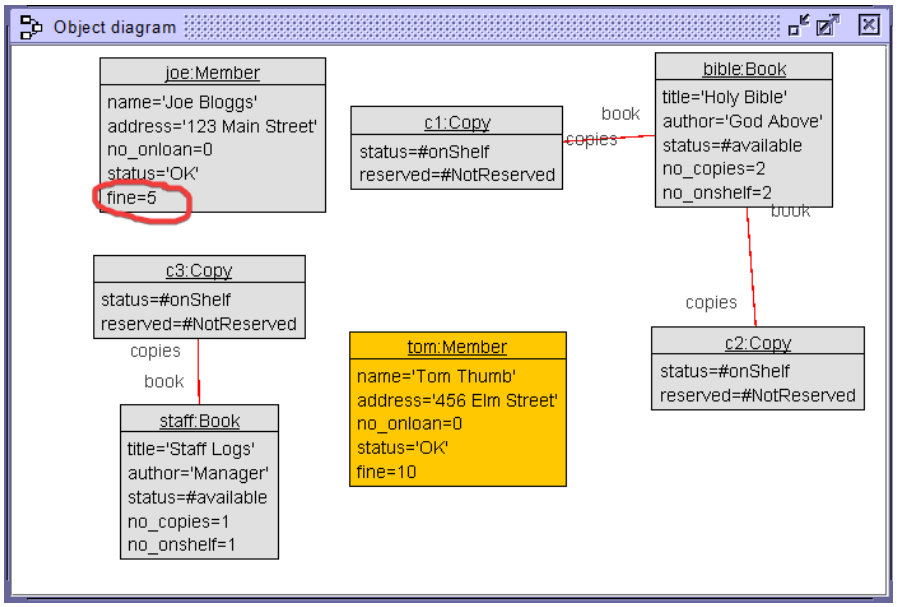
5. Pay Fine

Before:



After:

```
use> !joe.payFine(15)
use>
```

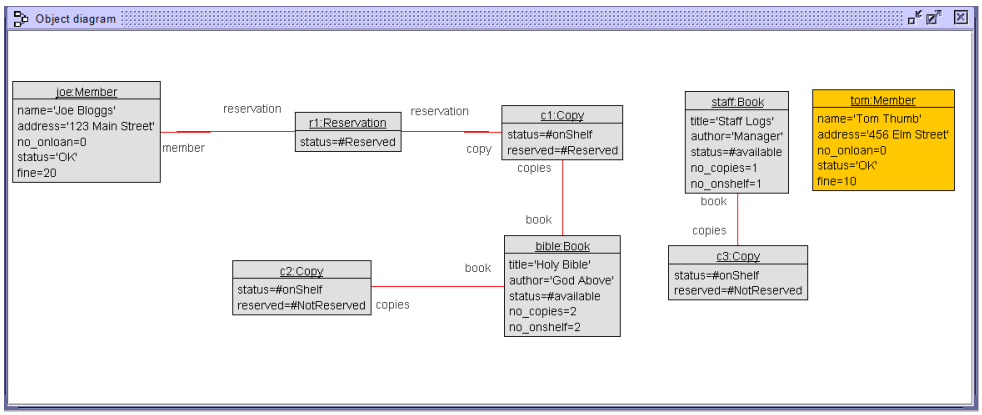


6. Testing constraints with !openter and !opexit

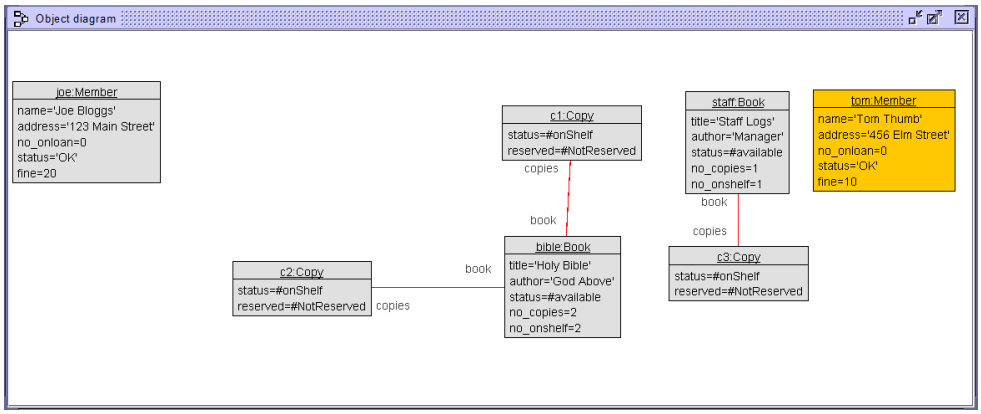
```
use> !joe.borrow(c1)
use> !openter joe renewLoan(c1)
precondition 'hasCopy' is true
precondition 'copyIsOnLoan' is true
use> !opexit
postcondition 'stillHasCopy' is true
use>
```

7. Testing post condition

Ensures that after calling removeReservation(), the reservation is completely removed.
Before:

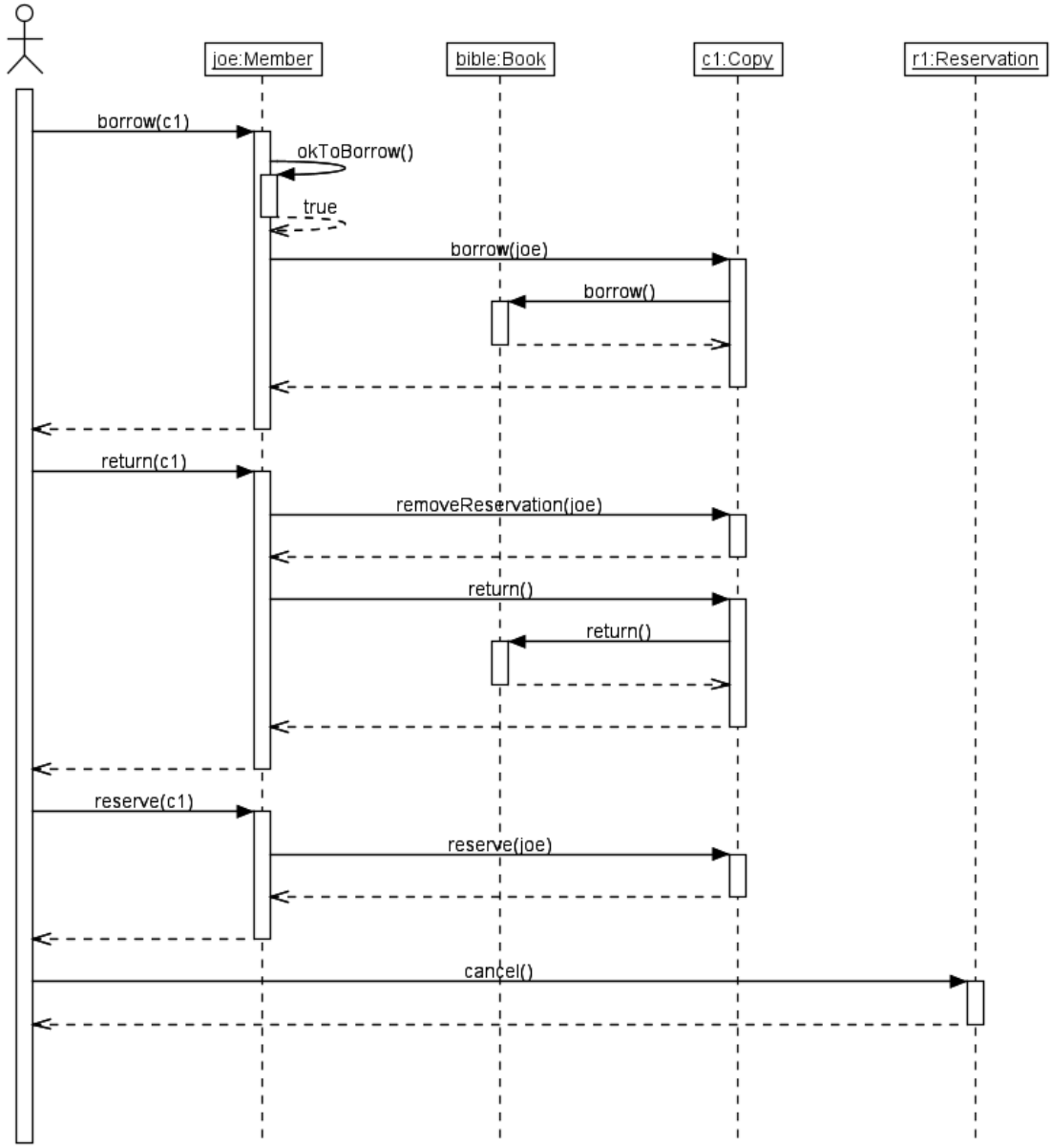


After:

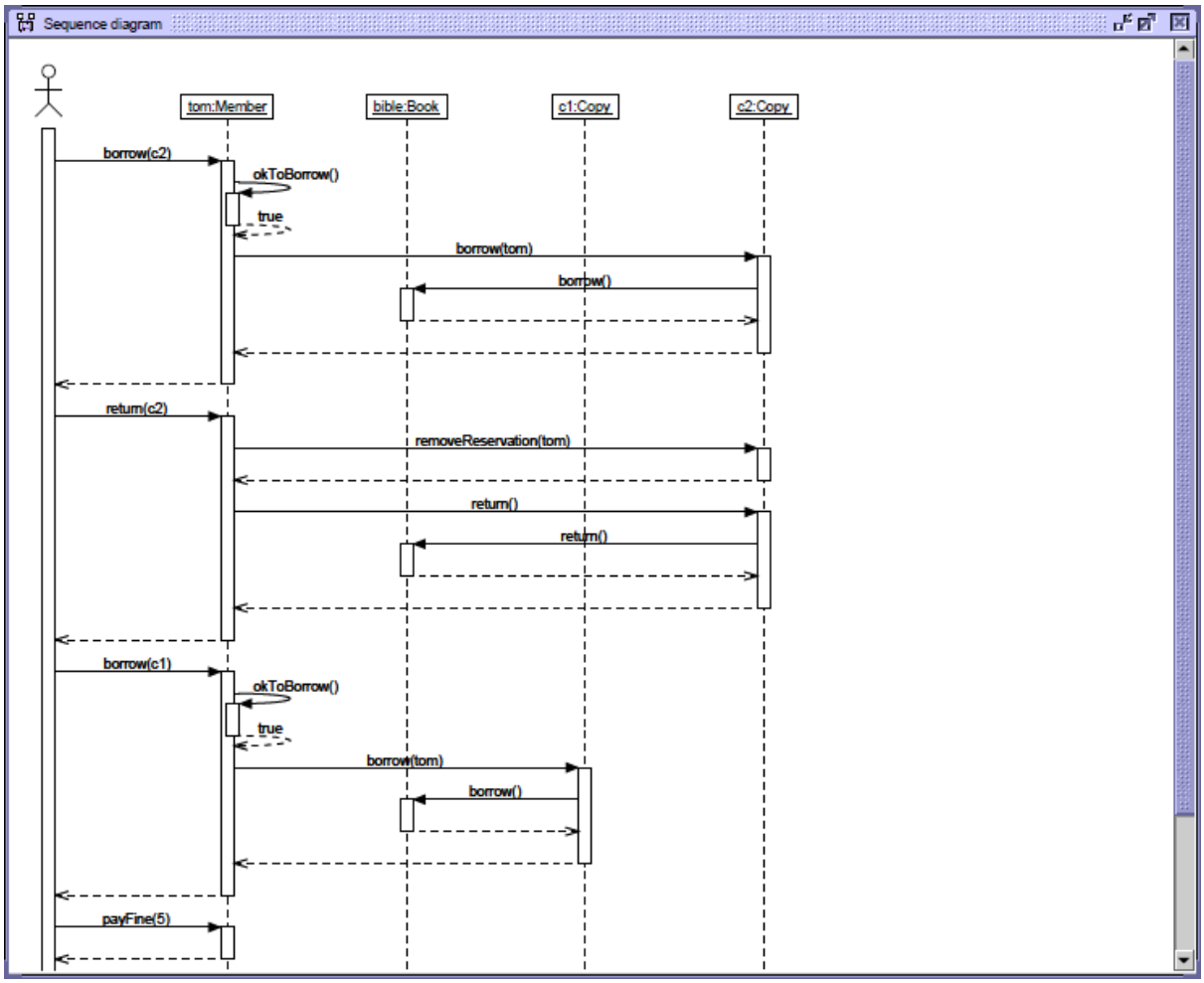

Object diagram
joe:Member
name='Joe Bloggs'
address='123 Main Street'
no_onloan=0
status='Ok'
fine=20
c1:Copy
status=#onShelf
reserved=#NotReserved
copies
book
staff:Book
title='Staff Logs'
author='Manager'
status=#available
no_copies=1
no_onshelf=1
book
copies
tom:Member
name='Tom Thumb'
address='456 Elm Street'
no_onloan=0
status='Ok'
fine=10
bible:Book
title='Holy Bible'
author='God Above'
status=#available
no_copies=2
no_onshelf=2
c2:Copy
status=#onShelf
reserved=#NotReserved
book
copies
c3:Copy
status=#onShelf
reserved=#NotReserved

Sequence Diagrams

Joe borrows a book, returns it, reserves a book, cancels a reservation.



Tom borrows a book, returns it, borrows another book, extends the loan on it and pays a fine



Conclusion

In this assignment I successfully designed and implemented extra use cases and the Reservation class and ensured the correct management of reserving, cancelling, paying fines and extending loans. Including these additional use cases not only added to the functionality of the system but also creagted a more realistic simulation of a library's operations.

The constraints I added included some business rules such as not being allowed to borrow the Staff Logs and handling reservations and cancellations. These constraints were tested to make sure they work correctly so that it can maintain the integrity of the library system. I made the Reservation class to manage the lifecycle of the reservations including whether a copy was reserved and making sure that they can be cancelled or modified.

The state machines play an important role in the lifecycle and status for the transitions of the books, copies and reservations. For example, the Book state machine tracks whether a book is available, unavailable or reserved, the Copy state machine manages the states of a copy such as it being on loan, on the shelf or reserved while the Reservation state machine tracks the status of the reservation. The state machines are a structured way to manage the state transitions of all the objects.

To show the system's functionality, I included sequence diagrams that clearly show how the Members, Joe and Tom, interact with the library system. The diagrams show the operations such as borrowing, reserving and returning books and how they flow as well as the fines and cancellations. They also show the functionality for extending a loan. This provides a visual representation of the sequence of the actions which help confirm the functionality of the design.

Using SOIL tests, I made sure the system works under different scenarios and that the post conditions were carefully checked to verify that reservations are removed after cancellation and that no lingering references exist in the system.

In conclusion, I have provided a comprehensive model of a library system while including the necessary features and testing the methods to make sure its correct and robust. I tested the system's functionality and made sure the functions were well-structured and working.