Software Engineering 2

Library Specification

Michael Lenghel
C16434974

Analysis of model:

For this assignment I modeled and extended a library which enables a user to borrow, reserve, cancel a reservation and return books. This model separates the implementation of reservations and puts it into its own class so that a reservation connects to a member and a member connects to a specific copy of a book.

The model uses appropriate pre and post conditions as well as invariants to prevent undesirable use of the system. The various states of the reservation system can be seen in the reservation class.

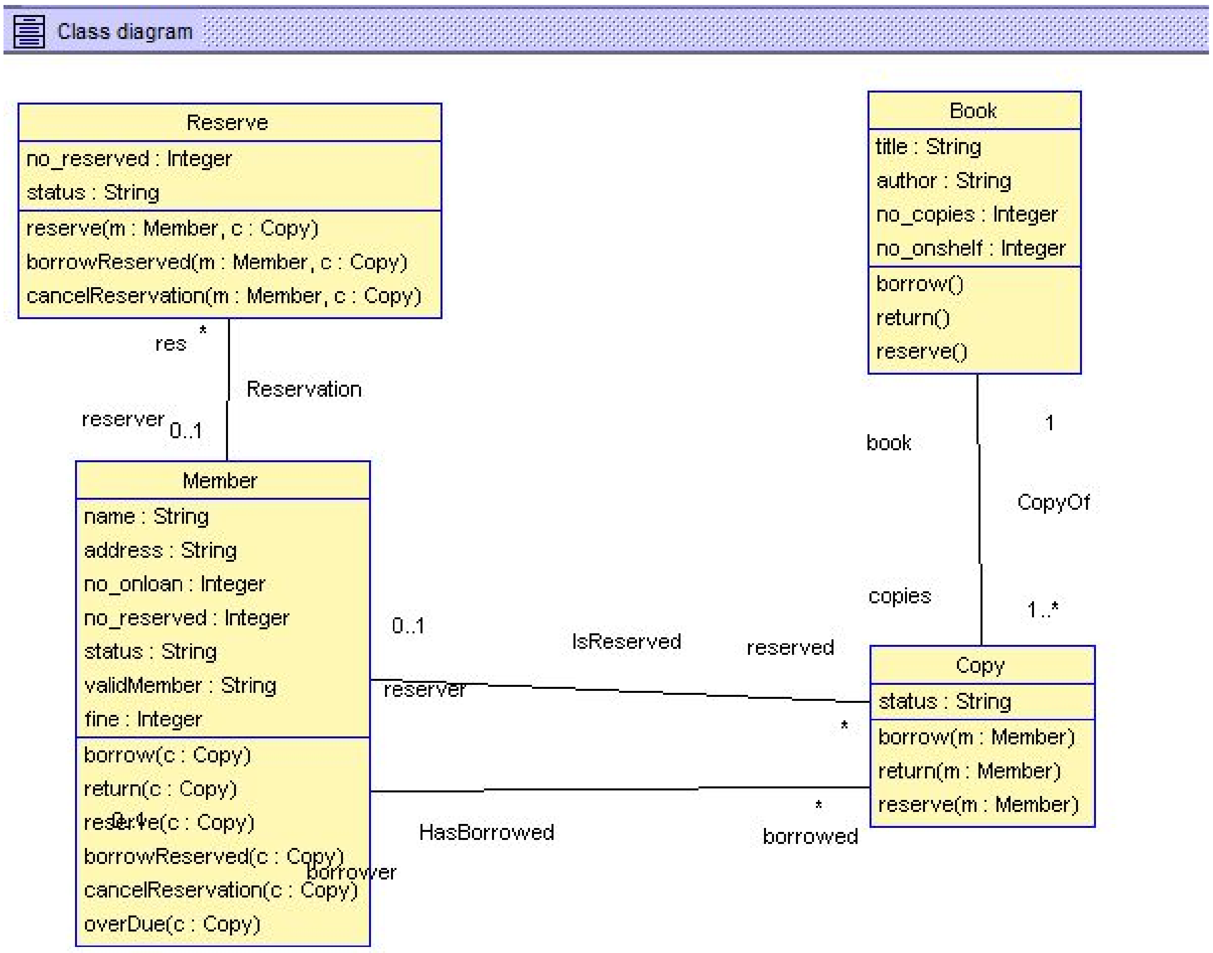
For the system, users cannot reserve or take out multiple copies of the same book.

Use Code:

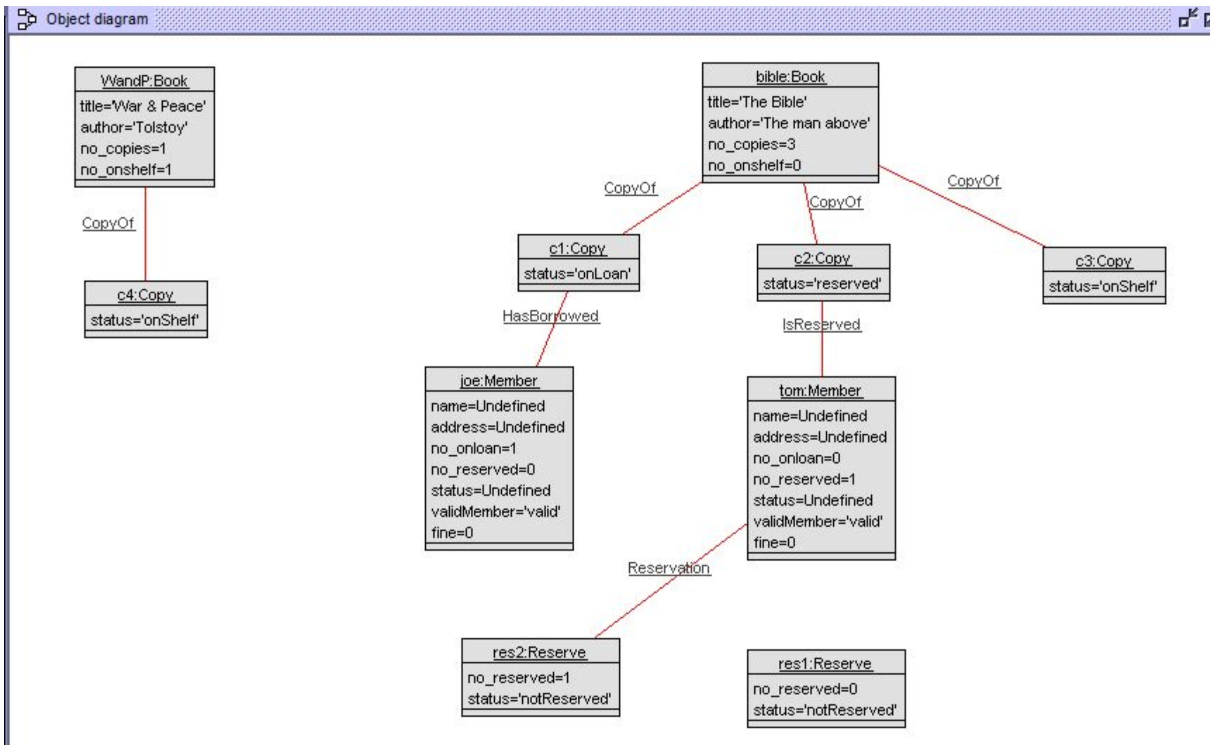
```
model Library

class Book
        attributes
                title : String
                author : String
                no_copies : Integer
                no_onshelf : Integer
        operations
                borrow()
                begin
                 self.no_onshelf := self.no_onshelf - 1
                end
                pre copiesOnShelf: no_copies > 0
                post: no_onshelf = no_onshelf@pre - 1

                return()
                begin
                        self.no_onshelf := self.no_onshelf + 1
                end
                post: no_onshelf = no_onshelf@pre + 1

                reserve()
                begin
                        self.no_onshelf := self.no_onshelf - 1
                end
                pre copiesOnShelf: no_copies > 0
end
class Copy
        attributes
        status : String init = 'onShelf'
        operations
                borrow( m : Member)
                begin
                self.status := 'onLoan';
                self.book.borrow()
                end

                return( m : Member)
                begin
                        self.status := 'onShelf';
                        self.book.return()
                end

                reserve( m : Member)
                begin
                        self.status := 'reserved';
                        self.book.reserve()
                end
end

class Reserve
        attributes
                no_reserved : Integer init = 0
        status : String init = 'notReserved'

        operations
        reserve(m : Member, c : Copy)
                begin
                        insert (m, self) into Reservation;
                        self.no_reserved := self.no_reserved + 1;
                        if (self.no_reserved = 1) then
                                self.status := 'reserved';
                                m.reserve(c)
                        end
                end

                borrowReserved(m : Member, c : Copy)
                begin
                        self.status := 'availableToReserve';
                        self.no_reserved := self.no_reserved - 1;
                        delete (m, self) from Reservation;
                        m.borrowReserved(c)
                end

                cancelReservation(m : Member, c : Copy)
                begin
                        self.status := 'availableToReserve';
                        delete (m, self) from Reservation;
                        self.no_reserved := self.no_reserved - 1;
                        --We can call return and not an independant cancel fxn as it's contents
are the exact same
                        m.cancelReservation(c)
                end

        --Statemachine which will show how the state fluctuates in the reservation classes
        statemachines
        psm States
        states
                newReservation : initial
                availableToReserve [no_reserved = 0]
                reserved [no_reserved > 0]
        transitions
                newReservation -> availableToReserve { create }
                availableToReserve -> reserved { [no_reserved = 0] reserve() }
                availableToReserve -> availableToReserve { cancelReservation() }
                reserved -> availableToReserve { cancelReservation() }
                availableToReserve -> availableToReserve { borrowReserved() }
                reserved -> availableToReserve { borrowReserved() }
        end
end

class Member
        attributes
                name : String
                address : String
                no_onloan : Integer init = 0
                no_reserved : Integer init = 0
                status : String
                --Memebrship can expire
                validMember : String init = 'valid'
                fine : Integer init = 0
        operations
                borrow(c : Copy)
                begin
                        insert (self, c) into HasBorrowed;
                        self.no_onloan := self.no_onloan + 1;
                        c.borrow(self)
                end

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

                reserve(c : Copy)
                begin
                        insert (self, c) into IsReserved;
                        self.no_reserved := self.no_reserved + 1;
                        c.reserve(self)
                end
        borrowReserved(c : Copy)
        begin
            self.no_reserved := self.no_reserved - 1;
            delete (self, c) from IsReserved;
            insert (self, c) into HasBorrowed;
            self.no_onloan := self.no_onloan + 1;
            c.borrow(self)
        end

        cancelReservation(c : Copy)
        begin
            delete (self, c) from IsReserved;
            self.no_reserved := self.no_reserved - 1;
            --We can call return and not an independant cancel fxn as it's contents
are the exact same
            c.return(self)
        end

        overDue(c : Copy)
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 IsReserved between
    Member[0..1] role reserver
    Copy[*] role reserved
end

association Reservation between
    Member[0..1] role reserver
    Reserve[*] role res
end

constraints
context Member::borrow(c:Copy)
    pre limit: self.no_onloan < 1
    pre cond1: self.borrowed->excludes(c)
    pre borrowIfAvailable: c.status = 'onShelf'
    post cond2: self.borrowed->includes(c)
```

```
context Member::return(c:Copy)
	pre cond1: self.borrowed->includes(c)
	post cond2: self.borrowed->excludes(c)

context Member::overDue(c:Copy)
	--Must be borrowed to be overdue
	pre cond1: self.borrowed->includes(c)
	--Can't already be overdue
	post setFine: self.fine > 0

context Member::reserve(c:Copy)
--Make sure that no connection between memebr and copy
pre cond1: self.reserved->excludes(c)
pre NoReserveSameCopy: self.no_reserved < 1
--Make sure member and copy are connected
post cond2: self.reserved->includes(c)
--Only can reserve if available
pre borrowIfAvailable: c.status = 'onShelf'

context Reserve::reserve(m:Member, c:Copy)
	--Make sure no connection between reservation and member
	pre cond1: self.reserver->excludes(m)
	--Only can reserve if available
	pre borrowIfAvailable: c.status = 'onShelf'
	--Only can reserve if has no overdue fees
	pre noOverDuebooks: m.status <> 'overDue'
	--Can only reserve 1 book
	pre limit: m.no_reserved < 1
	--Make reservation and member are connected
	post cond2: self.reserver->includes(m)

context Reserve::borrowReserved(m:Member, c:Copy)
	pre cond1: m.reserved->includes(c)
	post con2: m.reserved->excludes(c)

context Reserve::cancelReservation(m:Member, c:Copy)
	pre cond1: m.reserved->includes(c)
	post con2: m.reserved->excludes(c)



context Member inv: self.validMember = 'valid'
context Book inv: self.no_copies > -1
context Reserve inv: self.no_reserved > -1
--context Member inv: self.status <> 'overDue'
```

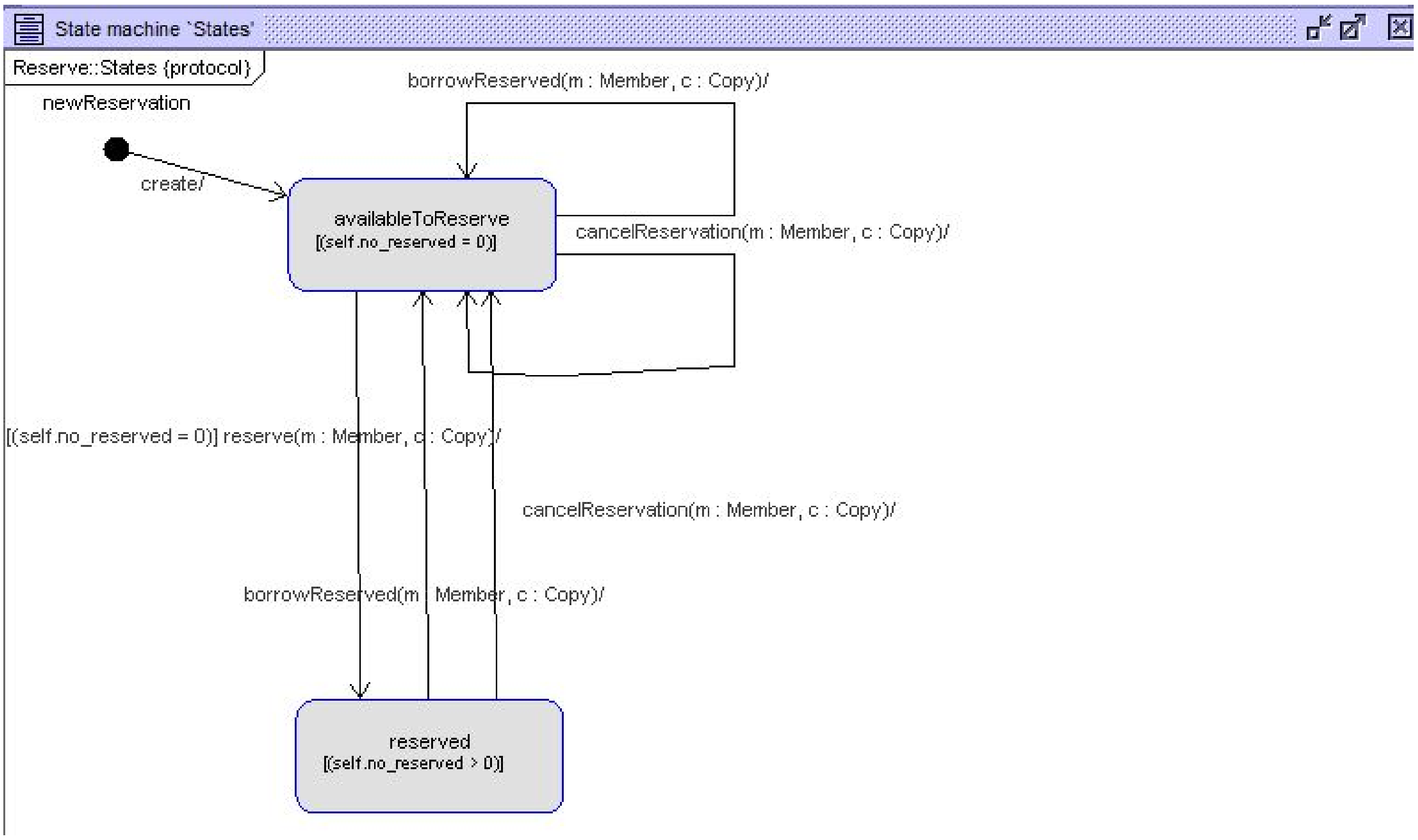
Class Diagram:



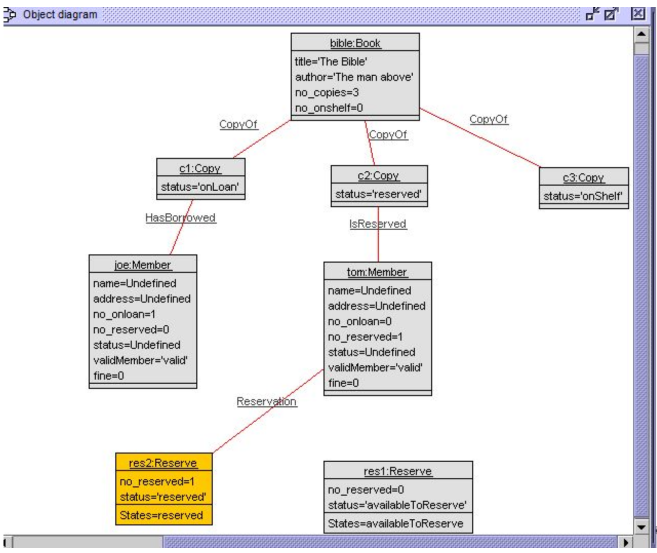
Object Diagram:

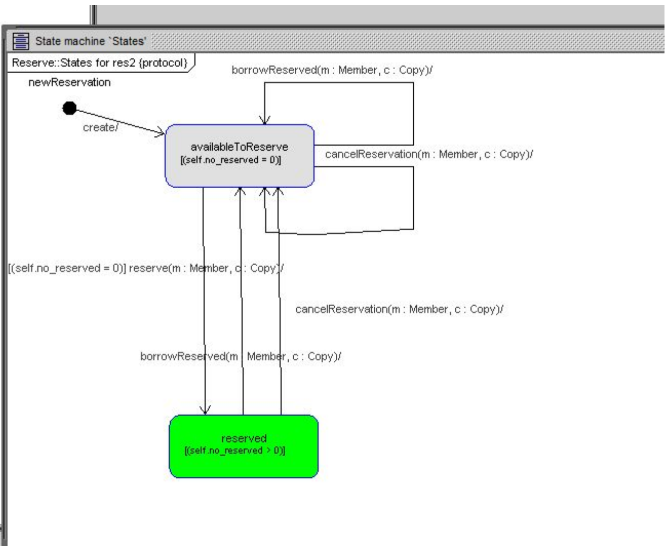


State Machine:



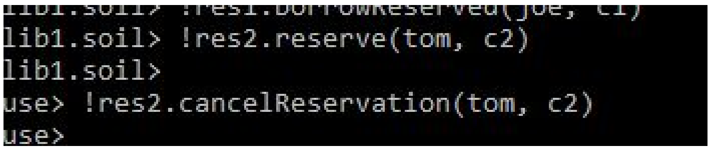
State machine `States'
Reserve::States {protocol}
newReservation
create/
borrowReserved(m : Member, c : Copy)/
availableToReserve
[(self.no_reserved = 0)]
cancelReservation(m : Member, c : Copy)/
[(self.no_reserved = 0)] reserve(m : Member, c : Copy)/
cancelReservation(m : Member, c : Copy)/
borrowReserved(m : Member, c : Copy)/
reserved
[(self.no_reserved > 0)]



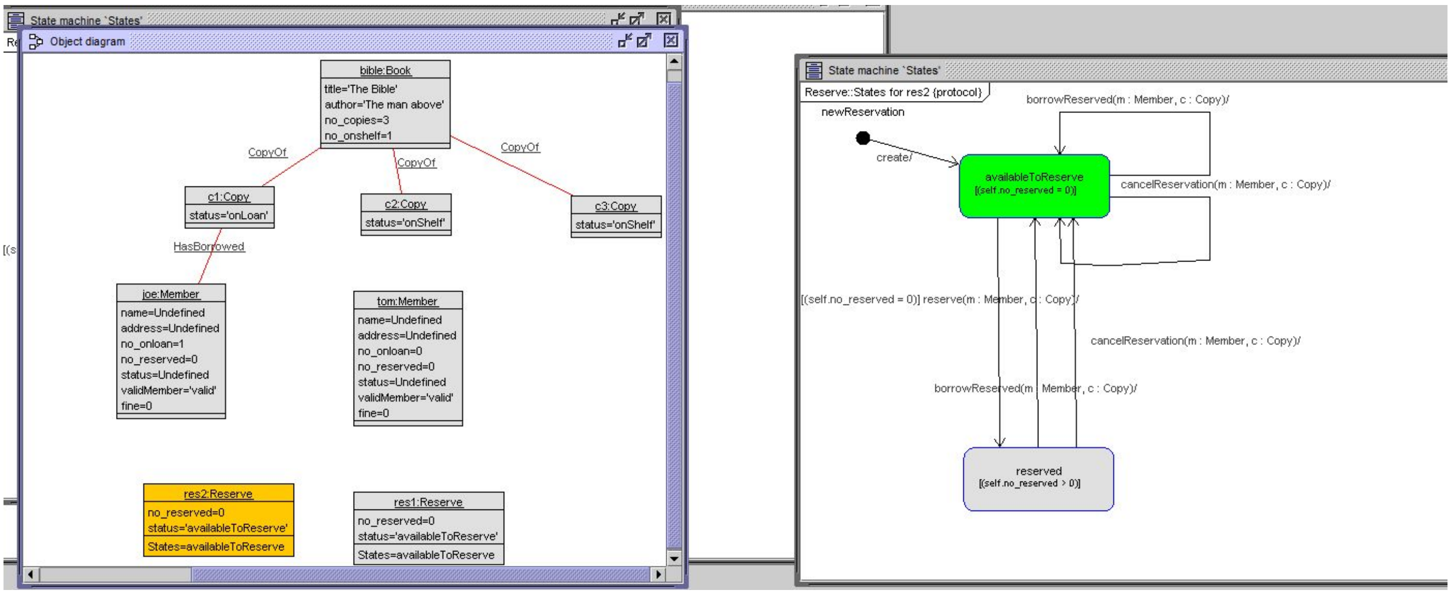


State machine `States'
Reserve::States for res2 {protocol}
newReservation
create/
borrowReserved(m : Member, c : Copy)/
availableToReserve
[(self.no_reserved = 0)]
cancelReservation(m : Member, c : Copy)/
[(self.no_reserved = 0)] reserve(m : Member, c : Copy)/
cancelReservation(m : Member, c : Copy)/
borrowReserved(m : Member, c : Copy)/
reserved
[(self.no_reserved > 0)]

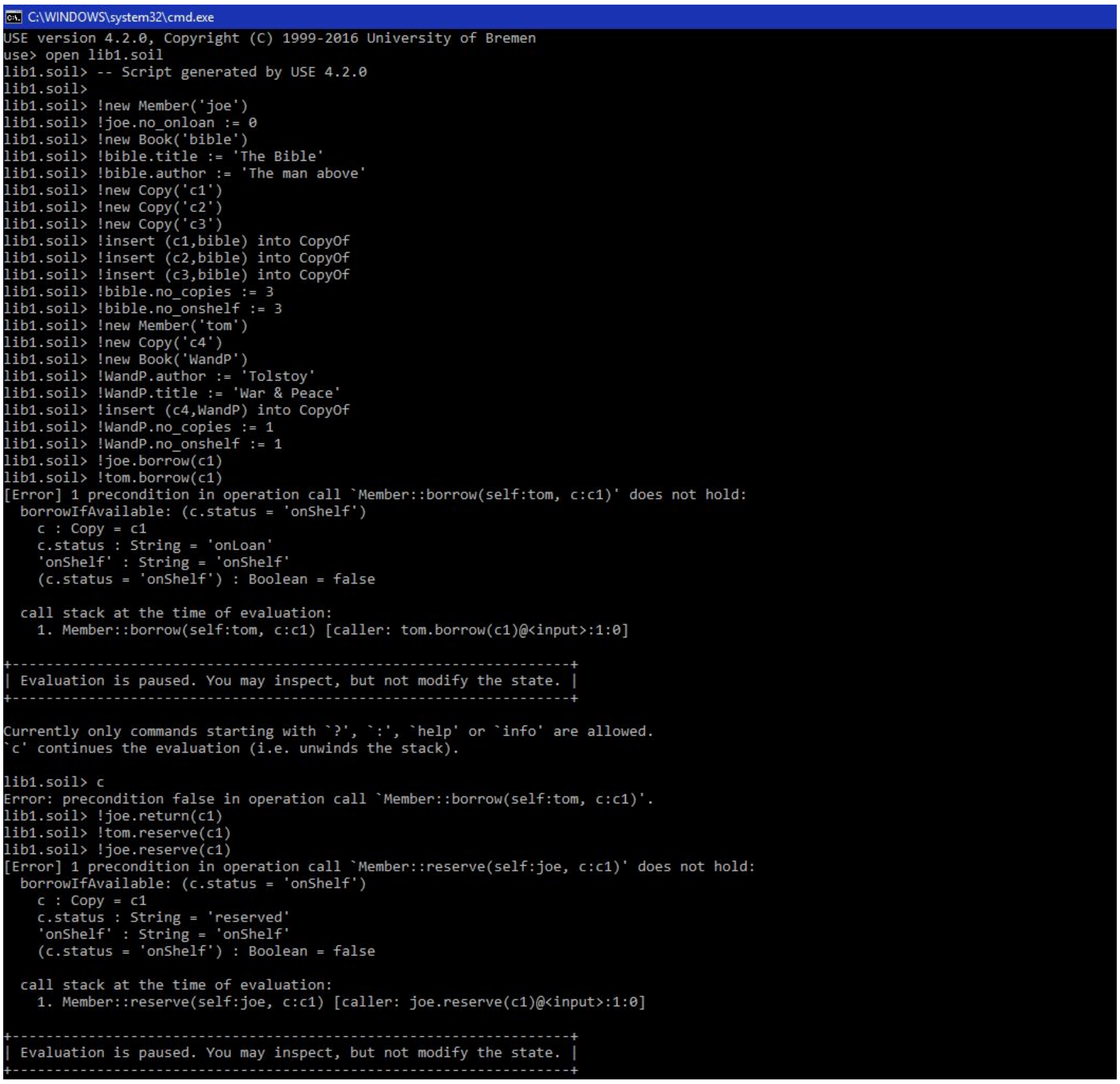
I canceled the reservation between Tom and copy 2 and this changed the state from reserved to available as seen below.



```
lib1.soil> !res1.borrowReserved(joe, c1)
lib1.soil> !res2.reserve(tom, c2)
lib1.soil>
use> !res2.cancelReservation(tom, c2)
use>
```



Command Prompt output:

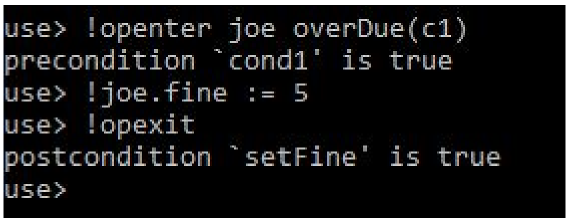


C:\WINDOWS\system32\cmd.exe

```
USE version 4.2.0, Copyright (C) 1999-2016 University of Bremen
use> open lib1.soil
lib1.soil> -- Script generated by USE 4.2.0
lib1.soil>
lib1.soil> !new Member('joe')
lib1.soil> !joe.no_onloan := 0
lib1.soil> !new Book('bible')
lib1.soil> !bible.title := 'The Bible'
lib1.soil> !bible.author := 'The man above'
lib1.soil> !new Copy('c1')
lib1.soil> !new Copy('c2')
lib1.soil> !new Copy('c3')
lib1.soil> !insert (c1,bible) into CopyOf
lib1.soil> !insert (c2,bible) into CopyOf
lib1.soil> !insert (c3,bible) into CopyOf
lib1.soil> !bible.no_copies := 3
lib1.soil> !bible.no_onshelf := 3
lib1.soil> !new Member('tom')
lib1.soil> !new Copy('c4')
lib1.soil> !new Book('WandP')
lib1.soil> !WandP.author := 'Tolstoy'
lib1.soil> !WandP.title := 'War & Peace'
lib1.soil> !insert (c4,WandP) into CopyOf
lib1.soil> !WandP.no_copies := 1
lib1.soil> !WandP.no_onshelf := 1
lib1.soil> !joe.borrow(c1)
lib1.soil> !tom.borrow(c1)
[Error] 1 precondition in operation call `Member::borrow(self:tom, c:c1)' does not hold:
  borrowIfAvailable: (c.status = 'onShelf')
    c : Copy = c1
    c.status : String = 'onLoan'
    'onShelf' : String = 'onShelf'
    (c.status = 'onShelf') : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:tom, c:c1) [caller: tom.borrow(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).

lib1.soil> c
Error: precondition false in operation call `Member::borrow(self:tom, c:c1)'.
lib1.soil> !joe.return(c1)
lib1.soil> !tom.reserve(c1)
lib1.soil> !joe.reserve(c1)
[Error] 1 precondition in operation call `Member::reserve(self:joe, c:c1)' does not hold:
  borrowIfAvailable: (c.status = 'onShelf')
    c : Copy = c1
    c.status : String = 'reserved'
    'onShelf' : String = 'onShelf'
    (c.status = 'onShelf') : Boolean = false

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

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

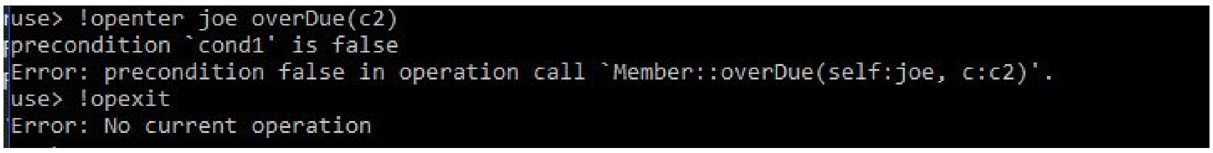
Openter and opexit commands

Openter / Opexit with precondition and postcondition set to true



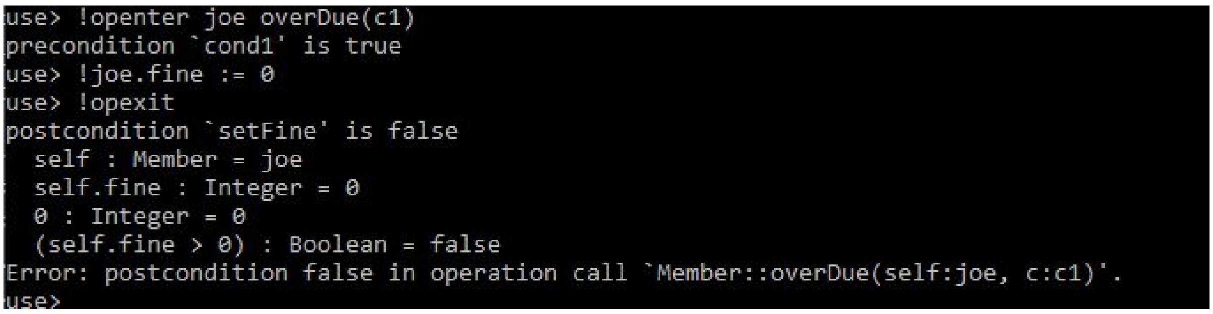
```
use> !openter joe overDue(c1)
precondition `cond1' is true
use> !joe.fine := 5
use> !opexit
postcondition `setFine' is true
use>
```

Openter / Opexit with precondition false and postcondition set to false



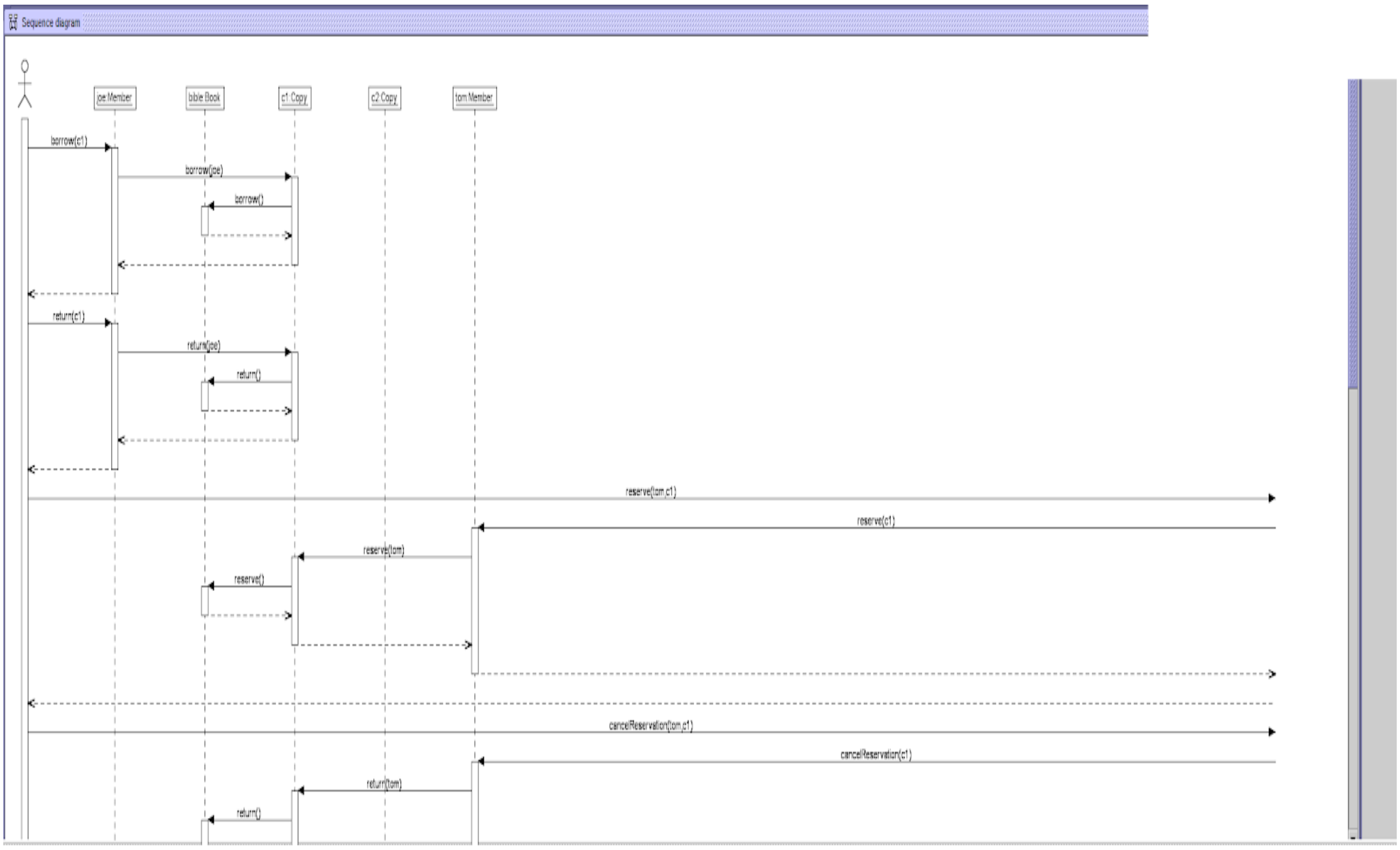
```
use> !openter joe overDue(c2)
precondition `cond1' is false
Error: precondition false in operation call `Member::overDue(self:joe, c:c2)'.
use> !opexit
Error: No current operation
```

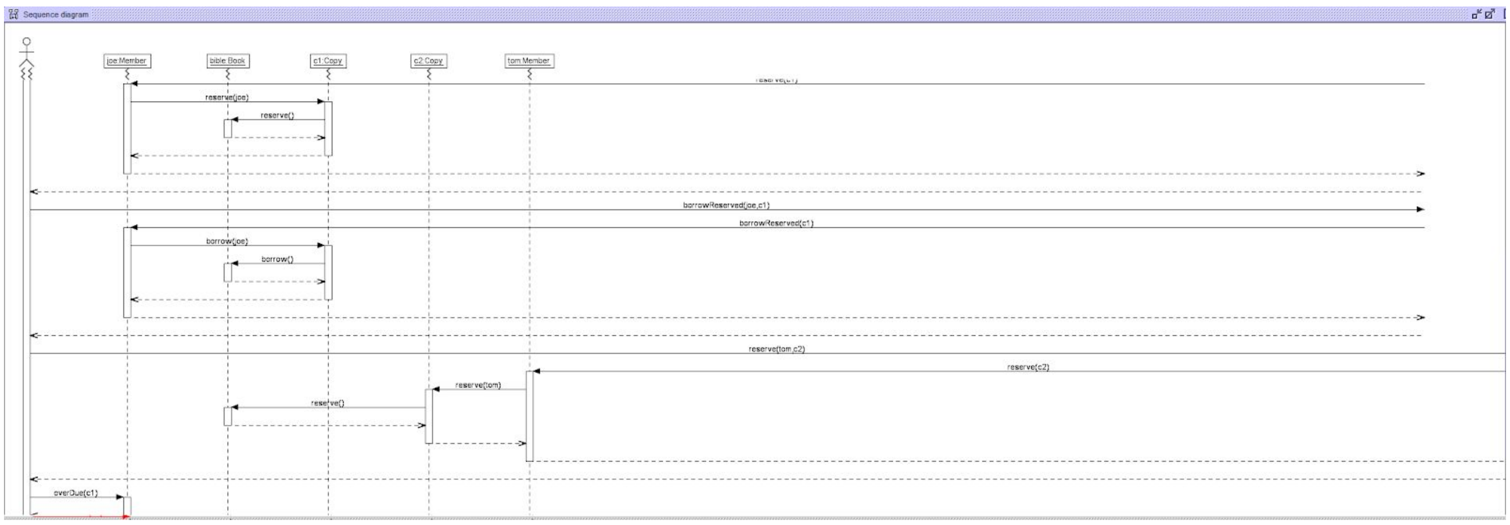
Openter with true precondition and false postcondition

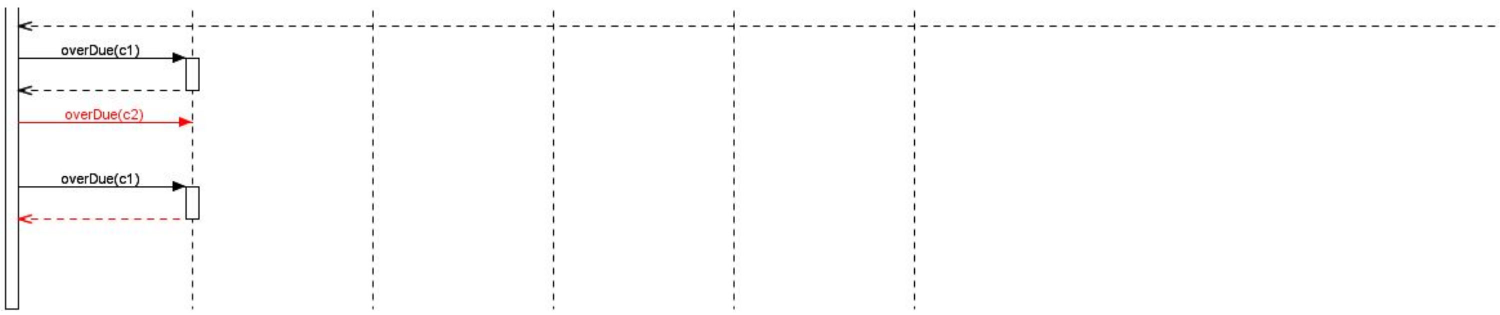


```
use> !openter joe overDue(c1)
precondition `cond1' is true
use> !joe.fine := 0
use> !opexit
postcondition `setFine' is false
  self : Member = joe
  self.fine : Integer = 0
  0 : Integer = 0
  (self.fine > 0) : Boolean = false
Error: postcondition false in operation call `Member::overDue(self:joe, c:c1)'.
use>
```

Sequence Diagrams:


Sequence diagram
joe:Member
bible:Book
c1:Copy
c2:Copy
tom:Member
borrow(c1)
borrow(joe)
borrow()
return(c1)
return(joe)
return()
reserve(tom,c1)
reserve(c1)
reserve(tom)
reserve()
cancelReservation(tom,c1)
cancelReservation(c1)
return(tom)
return()


Sequence diagram
joe:Member
bible:Book
c1:Copy
c2:Copy
tom:Member
reserve(c1)
reserve(joe)
reserve()
borrowReserved(joe,c1)
borrowReserved(c1)
borrow(joe)
borrow()
reserve(tom,c2)
reserve(c2)
reserve(tom)
reserve()
overDue(c1)


overDue(c1)
overDue(c2)
overDue(c1)