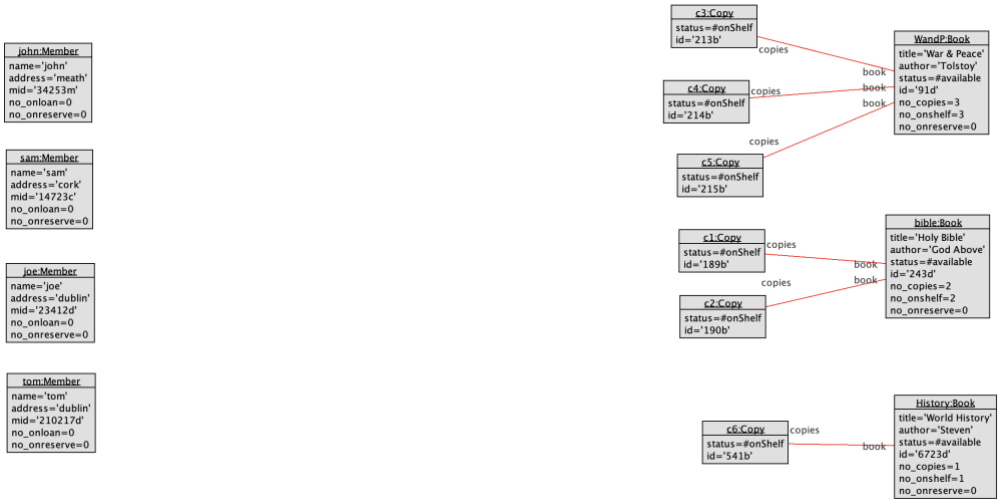
Software Engineering 2 Assignment Report

This is my report for the software engineering 2 assignment. I chose to extend and test a more comprehensive USE model for the library system in use. We came up with a plan on how to extend it and decided to extend the USE model by adding a reservation class and implementing the operations that come with that. We used the file Full-lib-with-sm-soln.use as a base file to start working and extend from. For all of the soil testing in my report, I used my soil file Library.soil to test the system. To create this soil file, I used the Full-lib.soil file as a base and added two more members, more copies and another book. Below is what the Library.soil file looks like in USE.

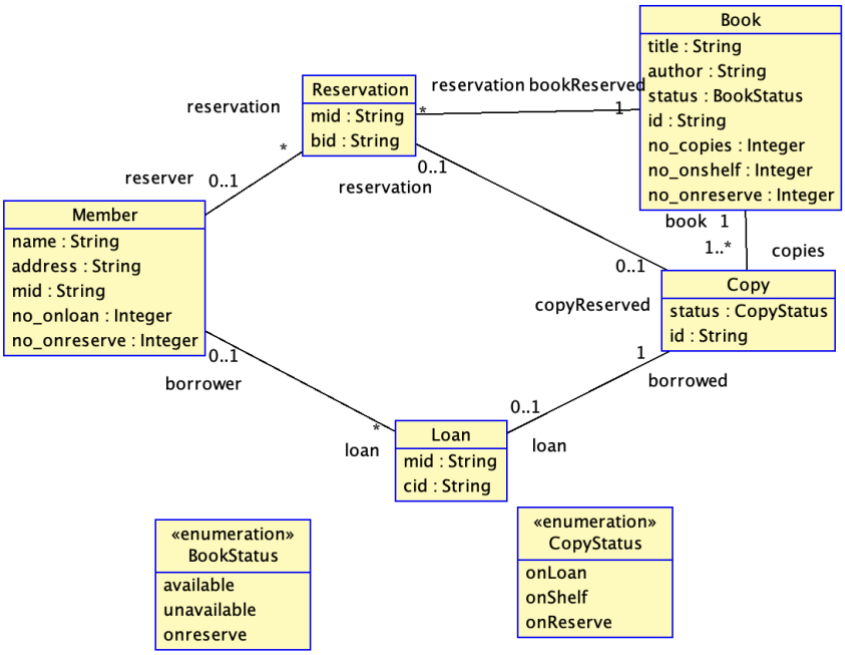


In this report I will look at the following:

1. Description of extra use case scenarios that were implemented
2. Preconditions, Postconditions and Invariants
3. Constraints tested with objects
4. Screen captures of testing on command line
5. Sequence Diagrams
6. State Machines
7. Object Diagrams
8. Class Diagrams
9. Discussion and Analysis
10. USE Code

Extra use case scenarios

The extra use case scenarios that we implemented to extend the library system were reserving a book, borrowing a reserved book and cancelling a reservation. All use case scenarios required many operations to change and new classes and operations to be made. While implementing these extra use case scenarios the library system changed quite a bit. We added a loan class that is an intermediatory class between the member, the book and copy they have on loan. We added a reservation class which we used as an intermediatory class between member and the book they have reserved. The member, copy and book class that were already implemented also changed with many operations being modified and new operations being added to them to implement these use case scenarios. The sequence diagrams for these use case diagrams can be seen further in the report. The object diagram below from USE shows all the classes and their associations.



Reserving a book

The first use case scenario I will describe is reserving a book. The idea here is that the user will be able to reserve a book if all copies are borrowed. This use case scenario begins in the member class with the reserve operation. It takes in a book object as an argument. It checks if the user is ok to reserve a book by calling the okToReserve operation which checks if the user has

less than 2 books on reserve. If they do it will return true and if they have 2 or more it returns false and then whatever is returned is stored in the variable called ok. It then checks if ok is true and if it is it will continue but if it is false the operation will end and the member will not be allowed to reserve a book. If the member is able to reserve a book a new reservation object, r, will be created and the book and member id will be set. The reservation and member are then inserted into the association between them called HasReserved and then the number of books they have on reserve is incremented by 1. The reserve operation then finally calls the newReservation operation from the reservation class with r passing in the member object and the book object as arguments. This operation begins by inserting the reservation and book object into the BookReserved association between them. It then loops through all the copies of the book and finds one that has its status set to onLoan. It then calls the reserve operation on the copy object. The reserve operation sets the copy status to onReserve and then calls the reserve operation on the book object associated with that copy. The reserve operation in the book class increases the amount on reserve by 1 and then checks if the number on reserve is equal to the number of copies there are and if so sets the book status to onreserve otherwise it ends. These operations will then keep returning back until they get back to newReservation operation in the reservation class where the member and copy are then inserted into the copyReserved association which is where the use case scenario then finishes.

Borrowing a reserved book

The second use case scenario I will describe is borrowing a reserved book. The scenario here is when the member has a book they have reserved and it has been returned by the member who was borrowing it so they can then borrow it. This use case scenario starts with the borrowReserved operation in the member class which takes in a reservation object as a argument. This operation begins by first checking if the member is ok to borrow by calling the okToBorrow operation. It is like the okToReserve operation but instead checks if the user has less then 2 books on loan it will return true and if they have 2 or more it will return false. If ok is true it will then continue and if it is false it will end and will not allow the member to borrow the book but the reservation will stay. If ok is true it will begin by calling the borrow operation from the reservation object that was passed in. The borrow operation in the reservation class begins by creating a new loan object and setting the copy and member id. It then calls the new loan objects

newLoanReserved operation and passes in the member who reserved the book and the copy that was reserved. The newLoanReserved operation inserts the member and copy into the CopyBorrowed association and then calls the copy's borrow operation and passes in the loan object. The copy's borrow operation then sets the copy's status to onLoan and checks if the copy has a reservation. In this scenario it does so it will call the borrowReserved operation for the book it is associated with but if it didn't, such as when just borrowing a book, it would call the borrow operation for the book. The book borrowReserved operation starts by reducing the number on reserve by 1. It then checks if the number on shelf is equal to 0 and if it is sets the status to unavailable otherwise it sets it to available. These operations then keep returning back until the borrow operation in the Reservation class, which then returns the new loan object back to the borrowReserved operation. This operation then destroys the reservation object, inserts the member and the new loan object that was returned into the HasBorrowed association and finally adds 1 to the number on loan and decreases the number on reserve by 1.

Cancelling a Reservation

The final use case scenario is cancelling a reservation. This scenario is if the member has a reservation and they want to cancel it. It begins with the cancelReservation operation in the member class which takes in a reservation object as an argument. This operation begins by first calling the book that was reserved cancelReservation operation and passing in the copy that was reserved as an argument. The cancelReservation operation in the book class checks if the copy is on loan by checking if the copy has an association with a loan. If it is not on loan it will decrease the number on reserve by 1 and set the status of the book to unavailable. If the copy is not on loan it increases the number of shelf by 1, decreases the number on reserve by 1 and then sets the status of the book to available. The operation then ends with calling the cancelReservation operation for the copy that was reserved. It checks if the copy is on loan the exact same way however this time if it is on loan it sets the copy status to onLoan and if it is on loan it sets the copy status to onShelf. These operations will then keep returning back to the cancelReservation operation in the member class. It will then decrease the number of books the member has on reserve by 1 and destroy the reservation object.

Preconditions, Postconditions and Invariants

Many preconditions, postconditions and invariants have been implemented into the system. For the extra use case scenarios, we specifically added preconditions and postconditions for the newLoanreserved operation in loan, the return, reserve, borrowReserved and cancelReservation operations in member. For preconditions, we have making sure the number the member has on loan is not greater than 2 in newLoanReserved, checking the number on shelf for the book is equal to 0 in the reserve operation and finally checking if the copy that the member has reserved is not on loan by anyone and checking if the status of the copy is onReserved in the borrowReserved operation. Alongside these preconditions we also have many that check if something is or is not in an association. For postconditions we only check if something is or is not in an association to confirm that the operationss have worked correctly for example member checks if the copy is now included in the association with loan. As well as all of these preconditions and postconditions for these extra use case scenarios, there is also ones for the return operation in book and the borrow operations in member.

For invariants, we added some to the member and book class. The member class has invariants for making sure the number on loan and the number on reserve is never negative. The book class has invariants that make sure the number of copies, the number on shelf and the number on reserve never go negative and makes sure the status stays as available, unavailable or onreserve.

Below is an image of the preconditions, postconditions and invariants in USE.

```
constraints


context Book::return()
    post bookReturnIncrease: no_onshelf = no_onshelf@pre + 1

context Member::borrow(c:Copy)
    pre memberBorrowLimit: self.no_onloan < 2
    pre memberStatusCheck: c.status = #onShelf
    pre memberBorrowExclusion: self.loan.borrowed->excludes(c)
    post memberBorrowInclusion: self.loan.borrowed->includes(c)

context Loan::newLoanReserved(m : Member, c : Copy)
    pre loanNewLoanResLimit: m.no_onloan < 2
    pre loanNewLoanResExclusion: self.borrowed->excludes(c)
    post loanNewLoanResInclusion: self.borrowed->includes(c)

context Member:: return(c:Copy)
    pre memberReturnExclusion: self.loan.borrowed->includes(c)
    post memberReturnInclusion: self.loan.borrowed->excludes(c)

context Member:: reserve(b : Book)
    pre memberReserveMinimum: b.no_onshelf = 0
    post memberReserveInclusion: self.reservation.bookReserved->includes(b)

context Member:: borrowReserved(r:Reservation)
    pre memberBorrowResEmpty: r.copyReserved.loan -> isEmpty()
    pre memberBorrowResStatus: r.copyReserved.status = #onReserve
    pre memberBorrowResInclusion: self.reservation -> includes(r)
    post memberBorrowResExclusion: self.reservation -> excludes(r)

context Member:: cancelReservation(r : Reservation)
    pre memberCancelResInclusion: self.reservation -> includes(r)
    post memberCancelResExclusion: self.reservation -> excludes(r)

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

context Member
    inv loanMin: self.no_onloan >= 0
    inv reserveMin: self.no_onreserve >= 0

context Book
    inv numBookCopies: self.no_copies >= 0
    inv numBookShelf: self.no_onshelf >= 0
    inv numBookReserve: self.no_onreserve >= 0
    inv BookStatusInv: self.status = #available or self.status = #unavailable or self.status = #onreserve
```

Constraints tested with objects

Openter and Opexit

Member:: reserve(b : Book)

```
use> !openter tom reserve(History)
precondition `memberReserveMinimum' is false
Error: precondition false in operation call `Member::reserve(self:tom, b:History)'.
use> !History.no_onshelf := 0
use> !openter tom reserve(History)
precondition `memberReserveMinimum' is true
use> !opexit
postcondition `memberReserveInclusion' is false
  self : Member = tom
  self.reservation : Set(Reservation) = Set{}
  self.reservation->collect($e : Reservation | $e.bookReserved) : Bag(Book) = Bag{}
  b : Book = History
  self.reservation->collect($e : Reservation | $e.bookReserved)->includes(b) : Boolean = false
Error: postcondition false in operation call `Member::reserve(self:tom, b:History)'.
use> !openter tom reserve(History)
precondition `memberReserveMinimum' is true
use> !new Reservation
use> !insert(tom,Reservation1) into HasReserved
use> !insert(Reservation1,History) into BookReserved
use> !insert(Reservation1,c6) into CopyReserved
use> !opexit
postcondition `memberReserveInclusion' is true
use>
```

Member:: borrowReserved(r:Reservation)

```
use> !new Reservation
use> !openter tom borrowReserved(Reservation1)
precondition `memberBorrowResEmpty' is true
precondition `memberBorrowResStatus' is false
precondition `memberBorrowResInclusion' is false
Error: precondition false in operation call `Member::borrowReserved(self:tom, r:Reservation1)'.
use> !insert (tom,Reservation1) into HasReserved
use> !insert (Reservation1,History) into BookReserved
use> !insert (Reservation1,c6) into CopyReserved
use> !openter tom borrowReserved(Reservation1)
precondition `memberBorrowResEmpty' is true
precondition `memberBorrowResStatus' is false
precondition `memberBorrowResInclusion' is true
Error: precondition false in operation call `Member::borrowReserved(self:tom, r:Reservation1)'.
use> !c6.status := #onReserve
use> !openter tom borrowReserved(Reservation1)
precondition `memberBorrowResEmpty' is true
precondition `memberBorrowResStatus' is true
precondition `memberBorrowResInclusion' is true
use> !opexit
postcondition `memberBorrowResExclusion' is false
  self : Member = tom
  self.reservation : Set(Reservation) = Set{Reservation1}
  r : Reservation = Reservation1
  self.reservation->excludes(r) : Boolean = false
Error: postcondition false in operation call `Member::borrowReserved(self:tom, r:Reservation1)'.
use> !new loan
<input>:1:0: Expected type name, found `loan'.
use> !new Loan
use> !insert(joe,Loan1) into HasBorrowed
use> !insert(Loan1,c6) into CopyBorrowed
use> !openter tom borrowReserved(c6)
<input>:1:0: Type mismatch for operation borrowReserved(r : Reservation) in argument 1. Expected type `Reservation', found `Copy'.
use> !openter tom borrowReserved(Reservation1)
precondition `memberBorrowResEmpty' is false
precondition `memberBorrowResStatus' is true
precondition `memberBorrowResInclusion' is true
Error: precondition false in operation call `Member::borrowReserved(self:tom, r:Reservation1)'.
use> !destory Loan1
<input>:line 1:8 extraneous input 'Loan1' expecting EOF
use> !destroy Loan1
use> !openter tom borrowReserved(Reservation1)
precondition `memberBorrowResEmpty' is true
precondition `memberBorrowResStatus' is true
precondition `memberBorrowResInclusion' is true
use> !destroy Reservation1
use> !opexit
postcondition `memberBorrowResExclusion' is true
use>
```

Loan::newLoanReserved(m : Member, c : Copy)

```
use> !new Loan
use> !openter Loan1 newLoanReserved(tom,c6)
precondition `loanNewLoanResLimit' is true
precondition `loanNewLoanResExclusion' is true
use> !opexit
postcondition `loanNewLoanResInclusion' is false
  self : Loan = Loan1
  self.borrowed : OclVoid = null
  self.borrowed : Set(Copy) = Set{}
  c : Copy = c6
  self.borrowed->includes(c) : Boolean = false
Error: postcondition false in operation call `Loan::newLoanReserved(self:Loan1, m:tom, c:c6)'.
use> !tom.no_onloan := 3
use> !tom.no_onloan := 2
use> !insert (tom,Loan1) into HasBorrowed
use> !insert (Loan1,c6) into CopyBorrowed
use> !openter Loan2 newLoanReserved(tom,c6)
<input>:1:0: Variable `Loan2' in expression `Loan2' is undefined.
use> !openter Loan1 newLoanReserved(tom,c6)
precondition `loanNewLoanResLimit' is false
precondition `loanNewLoanResExclusion' is false
Error: precondition false in operation call `Loan::newLoanReserved(self:Loan1, m:tom, c:c6)'.
use> !destroy Loan1
use> !tom.no_onloan := 0
use> !new Loan
use> !openter Loan2 newLoanReserved
<input>:line 1:0 no viable alternative at input 'openter'
use> !openter Loan2 newLoanReserved(tom,c6)
precondition `loanNewLoanResLimit' is true
precondition `loanNewLoanResExclusion' is true
use> !insert (tom,Loan2) into HasBorrowed
use> !insert (Loan2,c6) into CopyBorrwe
<input>:1:0: Association `CopyBorrwe' does not exist.
use>
use> !insert Loan2,c6) into CopyBorrowed
<input>:line 1:0 no viable alternative at input 'insert'
use> !insert (Loan2,c6) into CopyBorrowed
use> !opexit
postcondition `loanNewLoanResInclusion' is true
use>
```

Soil Implementation Testing

Member:: cancelReservation(r : Reservation)

```
use> !john.borrow(c6)
use> !tom.reserve(History)
use> !tom.cancelReservation(Reservation1)
use> !john.borrow(c2)
use> !sam.borrow(c1)
use> !joe.reserve(bible)
use> !tom.cancelReservation(Reservation2)
[Error] 1 precondition in operation call `Member::cancelReservation(self:tom, r:Reservation2)' does not hold:
  memberCancelResInclusion: self.reservation->includes(r)
    self : Member = tom
    self.reservation : Set(Reservation) = Set{}
    r : Reservation = Reservation2
    self.reservation->includes(r) : Boolean = false

  call stack at the time of evaluation:
    1. Member::cancelReservation(self:tom, r:Reservation2) [caller: tom.cancelReservation(Reservation2)@<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::cancelReservation(self:tom, r:Reservation2)'.
use> !joe.cancelReservation(Reservation2)
use>
```

Member::borrow(c:Copy)

```
use> !joe.borrow(c2)
[Error] 1 precondition in operation call `Member::borrow(self:joe, c:c2)' does not hold:
  memberStatusCheck: (c.status = CopyStatus::onShelf)
    c : Copy = c2
    c.status : CopyStatus = CopyStatus::onLoan
    CopyStatus::onShelf : CopyStatus = CopyStatus::onShelf
    (c.status = CopyStatus::onShelf) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:joe, c:c2) [caller: joe.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:joe, c:c2)'.
use> !joe.borrow(c4)
use> !joe.borrow(c5)
use> !joe.borrow(c6)
[Error] 1 precondition in operation call `Member::borrow(self:joe, c:c6)' does not hold:
  memberBorrowLimit: (self.no_onloan < 2)
    self : Member = joe
    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:joe, c:c6) [caller: joe.borrow(c6)@<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:joe, c:c6)'.
use> !joe.return(c5)
use> !joe.return(c4)
use>
```

```
use> !john.borrow(c6)
[Error] 1 postcondition in operation call `Member::borrow(self:john, c:c6)' does not hold:
  memberBorrowInclusion: self.loan->collect($e : Loan | $e.borrowed)->includes(c)
    self : Member = john
    self.loan : Set(Loan) = Set{Loan1}
    $e : Loan = Loan1
    $e.borrowed : OclVoid = null
    self.loan->collect($e : Loan | $e.borrowed) : Bag(Copy) = Bag{null}
    c : Copy = c6
    self.loan->collect($e : Loan | $e.borrowed)->includes(c) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:john, c:c6) [caller: john.borrow(c6)@<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).

Library.soil> Error: postcondition false in operation call `Member::borrow(self:john, c:c6)'.
use> c
Error: Unknown command `c'. Try `help'.
use>
```

```
use> !john.borrow(c6)
use> !john.borrow(c6)
[Error] 2 preconditions in operation call `Member::borrow(self:john, c:c6)' do not hold:
  memberStatusCheck: (c.status = CopyStatus::onShelf)
    c : Copy = c6
    c.status : CopyStatus = CopyStatus::onLoan
    CopyStatus::onShelf : CopyStatus = CopyStatus::onShelf
    (c.status = CopyStatus::onShelf) : Boolean = false

  memberBorrowExclusion: self.loan->collect($e : Loan | $e.borrowed)->excludes(c)
    self : Member = john
    self.loan : Set(Loan) = Set{Loan1}
    $e : Loan = Loan1
    $e.borrowed : Copy = c6
    self.loan->collect($e : Loan | $e.borrowed) : Bag(Copy) = Bag{c6}
    c : Copy = c6
    self.loan->collect($e : Loan | $e.borrowed)->excludes(c) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:john, c:c6) [caller: john.borrow(c6)@<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:john, c:c6)'.
```

```
use> !joe.borrow(c6)
use> !joe.return(c6)
```

Member:: return(c:Copy)

```
use> !joe.return(c4)
[Error] 1 precondition in operation call `Member::return(self:joe, c:c4)' does not hold:
  memberReturnExclusion: self.loan->collect($e : Loan | $e.borrowed)->includes(c)
    self : Member = joe
    self.loan : Set(Loan) = Set{}
    self.loan->collect($e : Loan | $e.borrowed) : Bag(Copy) = Bag{}
    c : Copy = c4
    self.loan->collect($e : Loan | $e.borrowed)->includes(c) : Boolean = false

  call stack at the time of evaluation:
    1. Member::return(self:joe, c:c4) [caller: joe.return(c4)@<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::return(self:joe, c:c4)'.
use> !sam.return(c1)
use>
```

```
use> !joe.borrow(c4)
use> !joe.return(c4)
[Error] 1 postcondition in operation call `Member::return(self:joe, c:c4)' does not hold:
  memberReturnInclusion: self.loan->collect($e : Loan | $e.borrowed)->excludes(c)
    self : Member = joe
    self.loan : Set(Loan) = Set{Loan1}
    $e : Loan = Loan1
    $e.borrowed : Copy = c4
    self.loan->collect($e : Loan | $e.borrowed) : Bag(Copy) = Bag{c4}
    c : Copy = c4
    self.loan->collect($e : Loan | $e.borrowed)->excludes(c) : Boolean = false

  call stack at the time of evaluation:
    1. Member::return(self:joe, c:c4) [caller: joe.return(c4)@<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: postcondition false in operation call `Member::return(self:joe, c:c4)'.
use>
```

```
use> !joe.borrow(c4)
use> !joe.return(c4)
use>
```

Book::return()

```
use> !History.return()
[Error] 1 postcondition in operation call `Book::return(self:History)' does not hol
  bookReturnIncrease: (self.no_onshelf = (self.no_onshelf@pre + 1))
    self : Book = History
    self.no_onshelf : Integer = 1
    self : Book = History
    self.no_onshelf@pre : Integer = 1
    1 : Integer = 1
    (self.no_onshelf@pre + 1) : Integer = 2
    (self.no_onshelf = (self.no_onshelf@pre + 1)) : Boolean = false

  call stack at the time of evaluation:
    1. Book::return(self:History) [caller: History.return()@<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: postcondition false in operation call `Book::return(self:History)'.
 use> !History.return()
 use>
```

Screen Captures of Testing on Command Line

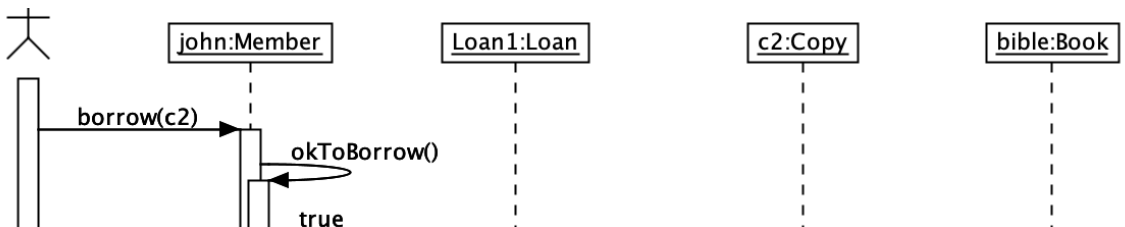
The table below is an overview of the testing I did. I tried to look at each scenario one by one and then decide what are the different possibilities in this scenario and see if the system works as expected. When I did my testing, the system passed and worked as expected in all scenarios.

Use-Case Scenario	Passed
Borrow a book	✓
Return a book	✓
Stops you from borrowing more than two books	✓
Reserve a book when there are copies available to borrow	✓
Reserve a book	✓
Cancel the reservation	✓
Returning a book that is reserved	✓
Borrow a Reserved book	✓
Borrow a Reserved book but already have two books borrowed	✓

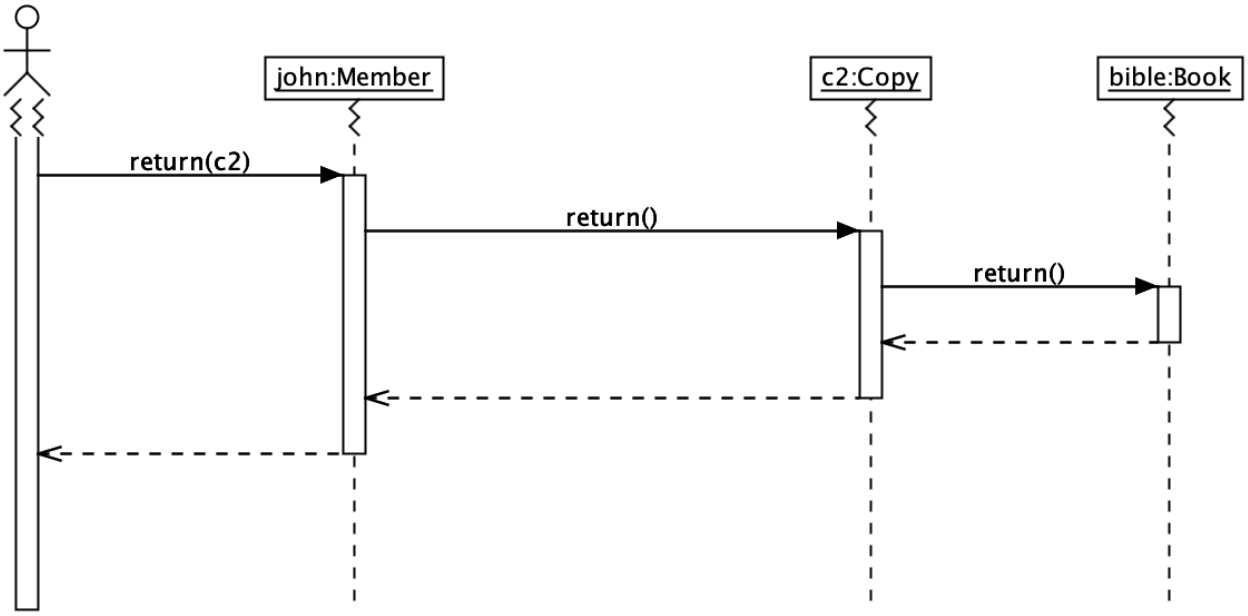
```
use> !john.borrow(c2)
use> !sam.borrow(c1)
use> !sam.borrow(c6)
use> !sam.borrow(c3)
[Error] 1 precondition in operation call `Member::borrow(self:sam, c:c3)' does not hold:
  memberBorrowLimit: (self.no_onloan < 2)
    self : Member = sam
    self.no_onloan : Integer = 2
    2 : Integer = 2
    (self.no_onloan < 2) : Boolean = false

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

> c
Error: precondition false in operation call `Member::borrow(self:sam, c:c3)'.
use> !joe.borrow(c6)
[Error] 1 precondition in operation call `Member::borrow(self:joe, c:c6)' does not hold:
  memberStatusCheck: (c.status = CopyStatus::onShelf)
    c : Copy = c6
    c.status : CopyStatus = CopyStatus::onLoan
    CopyStatus::onShelf : CopyStatus = CopyStatus::onShelf
    (c.status = CopyStatus::onShelf) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrow(self:joe, c:c6) [caller: joe.borrow(c6)@<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:joe, c:c6)'.
```

```
use> !joe.borrowReserved(Reservation3)
use> !sam.return(c1)
use> !joe.return(c2)
use> !tom.cancelReservation(Reservation4)
use> !joe.borrow(c1)
use> !tom.borrow(c2)
use> !john.reserve(bible)
use> !joe.return(c1)
use> !joe.borrow(c1)
use> !tom.reserve(bible)
use> !joe.return(c1)
use> !tom.return(c2)
use> !tom.cancelReservation(Reservation4)
<input>:1:0: Variable `Reservation4' in expression `<no string representation>' is undefined.
use> !tom.cancelReservation(Reservation6)
use> !john.borrowReserved(Reservation5)
use> !john.borrow(c3)
use> !sam.borrow(c6)
use> !john.reserve(History)
use> !sam.return(c6)
use> !john.borrowReserved(Reservation7)
[Error] 1 postcondition in operation call `Member::borrowReserved(self:john, r:Reservation7)' does not hold:
  memberBorrowResExclusion: self.reservation->excludes(r)
    self : Member = john
    self.reservation : Set(Reservation) = Set{Reservation7}
    r : Reservation = Reservation7
    self.reservation->excludes(r) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrowReserved(self:john, r:Reservation7) [caller: john.borrowReserved(Reservation7)@<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: postcondition false in operation call `Member::borrowReserved(self:john, r:Reservation7)'.
use> !john.return(c2)
use> 1john.borrowReserved(c6)
Error: Unknown command `1john.borrowReserved(c6)'. Try `help'.
use> !john.borrowReserved(c6)
<input>:1:0: Type mismatch for operation borrowReserved(r : Reservation) in argument 1. Expected type `Reservation', found `Copy'.
use> !john.borrowReserved(c6)
<input>:1:0: Type mismatch for operation borrowReserved(r : Reservation) in argument 1. Expected type `Reservation', found `Copy'.
use> !john.borrowReserved(Reservation7)
use> !sam.borrow(c4)
use> !joe.borrow(c5)
use> !tom.reserve(WandP)
use> !tom.reserve(WandP
<input>:line 1:0 no viable alternative at input 'tom'
use> !tom.reserve(WandP)
    > c
    Error: precondition false in operation call `Member::reserve(self:tom, b:History)'.
    use> !tom.cancelReservation(bible)
    <input>:1:0: Type mismatch for operation cancelReservation(r : Reservation) in argument 1. Expected type `Reservation', found `Book'.
    use> !tom.cancelReservation(Reservation2)
    use> !tom.reserve(bible)
    use> !john.return(c2)
    use> !joe.borrowReserved(Reservation4)
    [Error] 2 preconditions in operation call `Member::borrowReserved(self:joe, r:Reservation4)' do not hold:
      memberBorrowResEmpty: r.copyReserved.loan->isEmpty
        r : Reservation = Reservation4
        r.copyReserved : Copy = c1
        r.copyReserved.loan : Loan = Loan2
        r.copyReserved.loan : Set(Loan) = Set{Loan2}
        r.copyReserved.loan->isEmpty : Boolean = false

      memberBorrowResInclusion: self.reservation->includes(r)
        self : Member = joe
        self.reservation : Set(Reservation) = Set{Reservation3}
        r : Reservation = Reservation4
        self.reservation->includes(r) : Boolean = false

    call stack at the time of evaluation:
      1. Member::borrowReserved(self:joe, r:Reservation4) [caller: joe.borrowReserved(Reservation4)@<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::borrowReserved(self:joe, r:Reservation4)'.
```

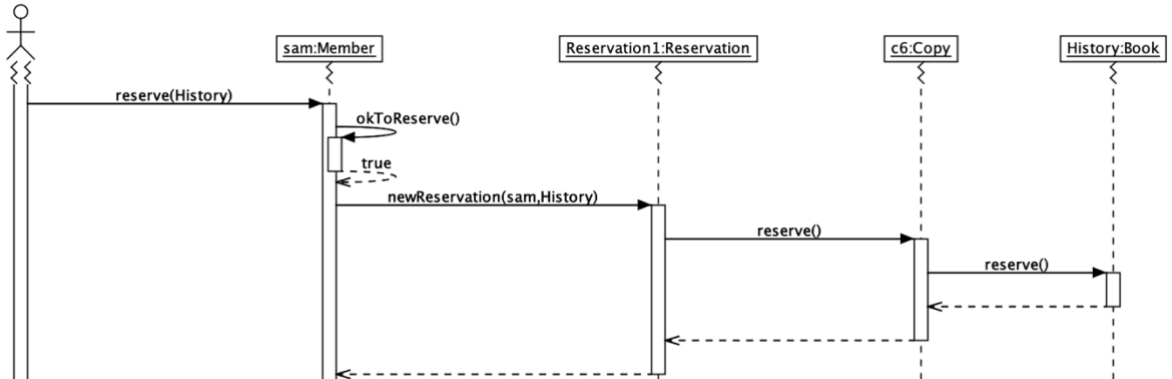
```
use> !joe.reserve(WandP)
use> !john.return(c3)
use> !sam.return(c4)
use> !tom.borrowReserved(Reservation9)
use> !tom.reserve(WandP)
use> !joe.cancelReservation(Reservation10)
use> !tom.cancelReservation(Reservation11)
use> !sam.reserve(WandP)
[Error] 1 precondition in operation call `Member::reserve(self:sam, b:WandP)' does not hold:
  memberReserveMinimum: (b.no_onshelf = 0)
    b : Book = WandP
    b.no_onshelf : Integer = 1
    0 : Integer = 0
    (b.no_onshelf = 0) : Boolean = false

  call stack at the time of evaluation:
    1. Member::reserve(self:sam, b:WandP) [caller: sam.reserve(WandP)@<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:sam, b:WandP)'.
use> !sam.borrow(c3)
use> !sam.reserve(WandP)
use> !sam.reserve(WandP)
use> !tom.return(c4)
use> !sam.borrowReserved(Reservation12)
use> !tom.reserve(History)
use> !john.return(c6)
use> !tom.borrowReserved(Reservation14)
use> !sam.return(c4)
use> !sam.return(c3)
use> !sam.cancelReservation(Reservation13)
use>
```



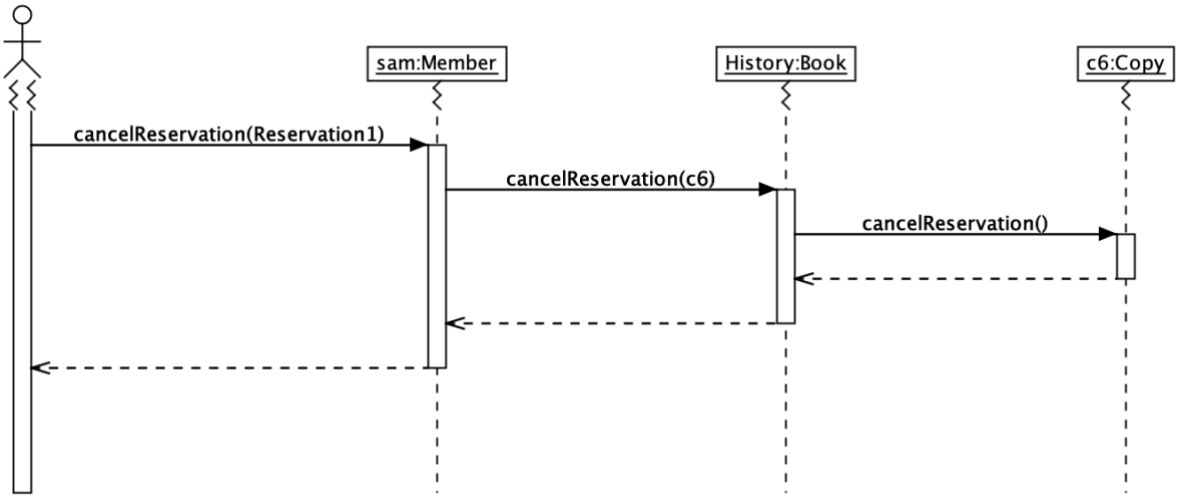
Returning a book



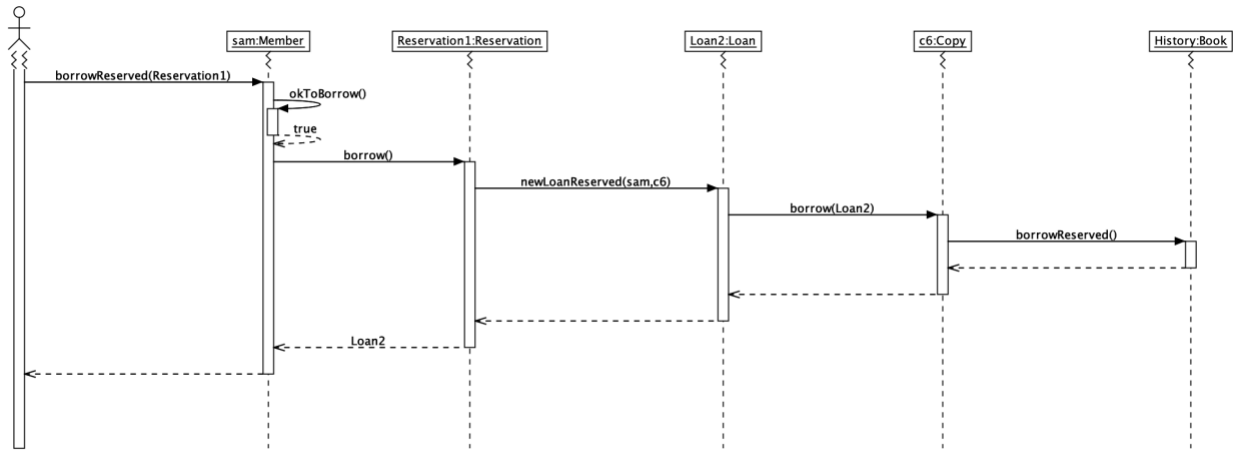
Reserving a book



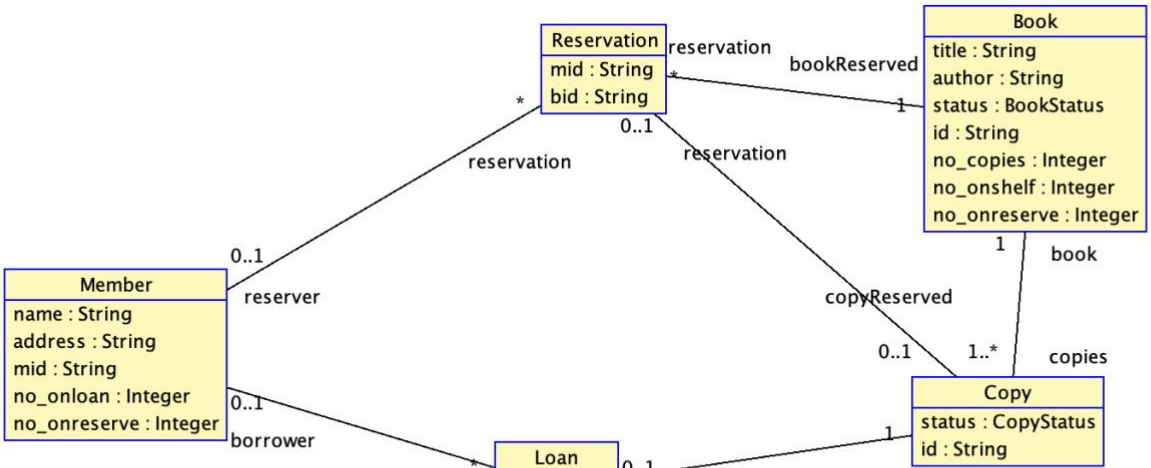
Cancelling a reservation

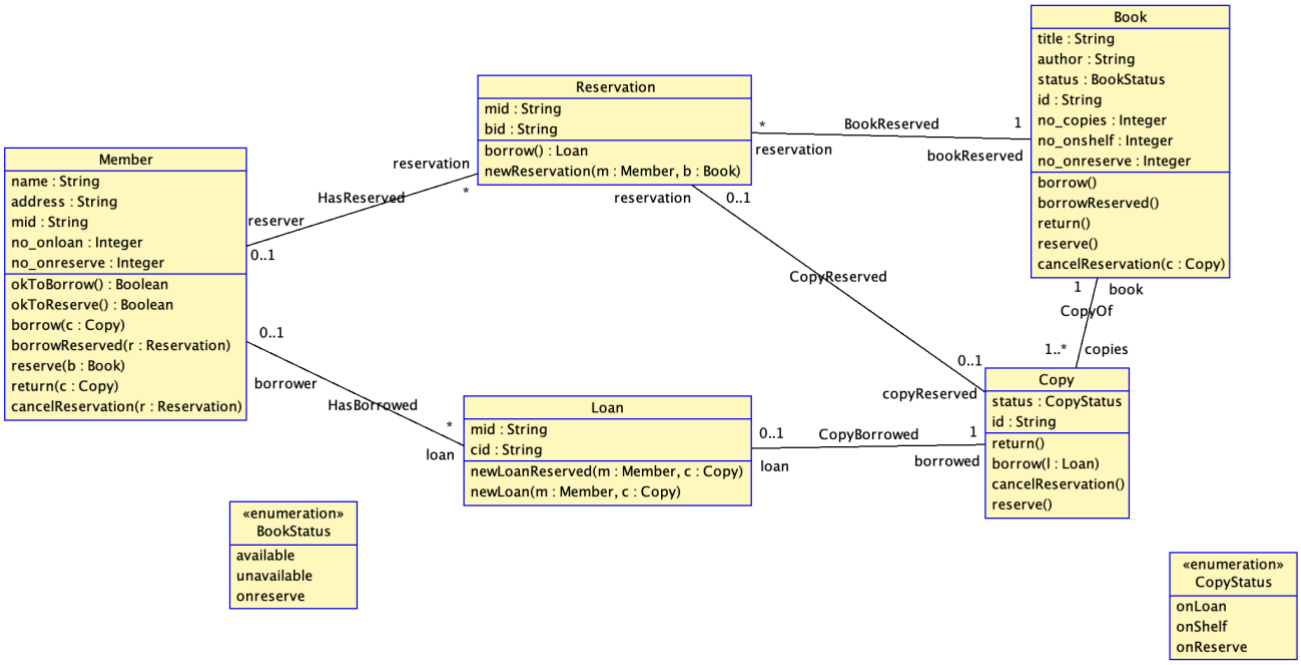


Borrowing a reserved book



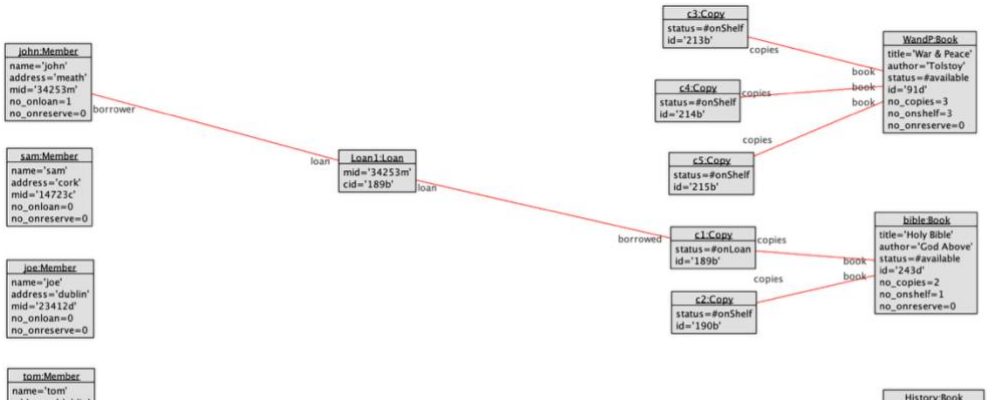
Class Diagrams

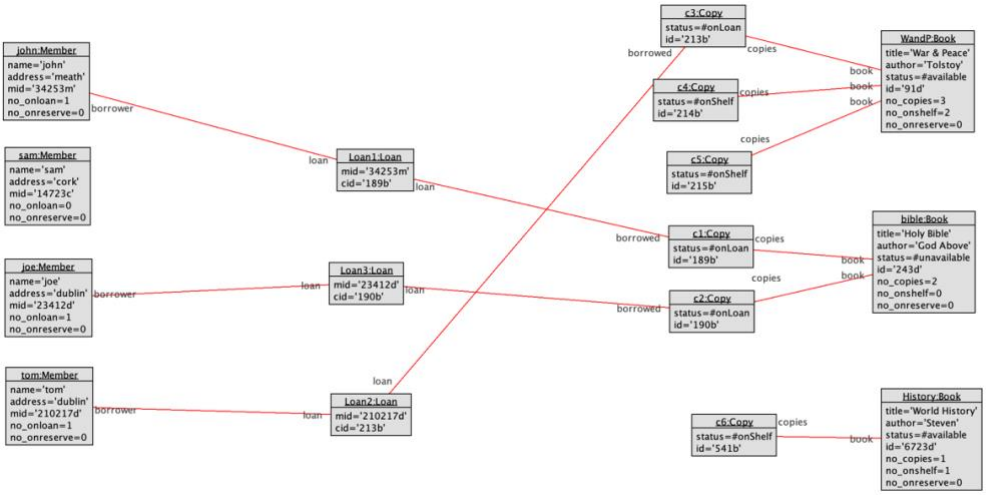


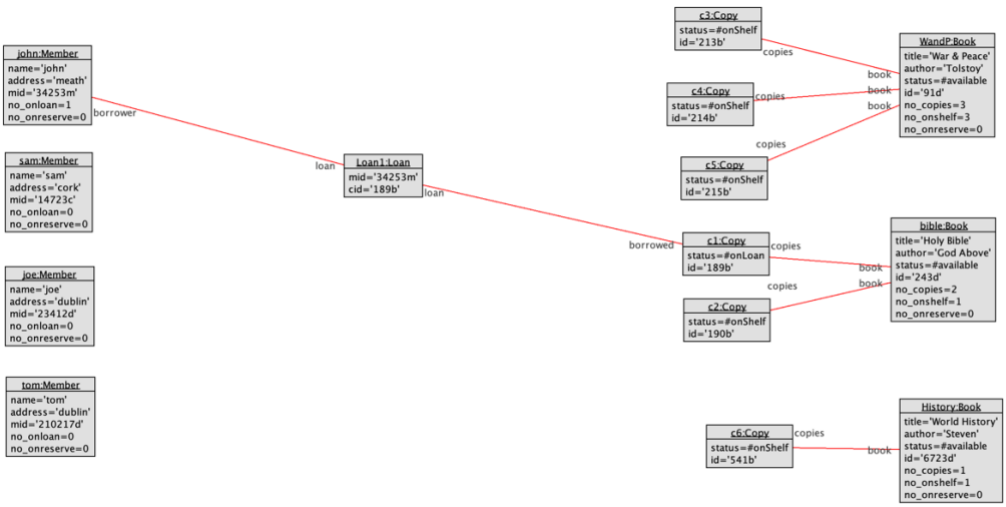


Object Diagrams

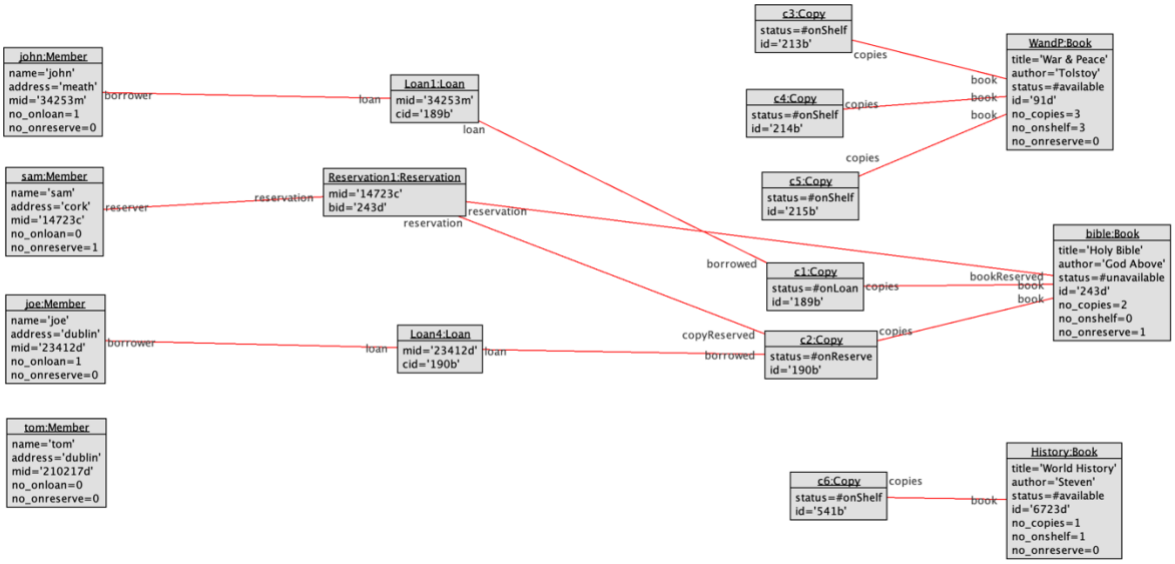
Borrowing a book

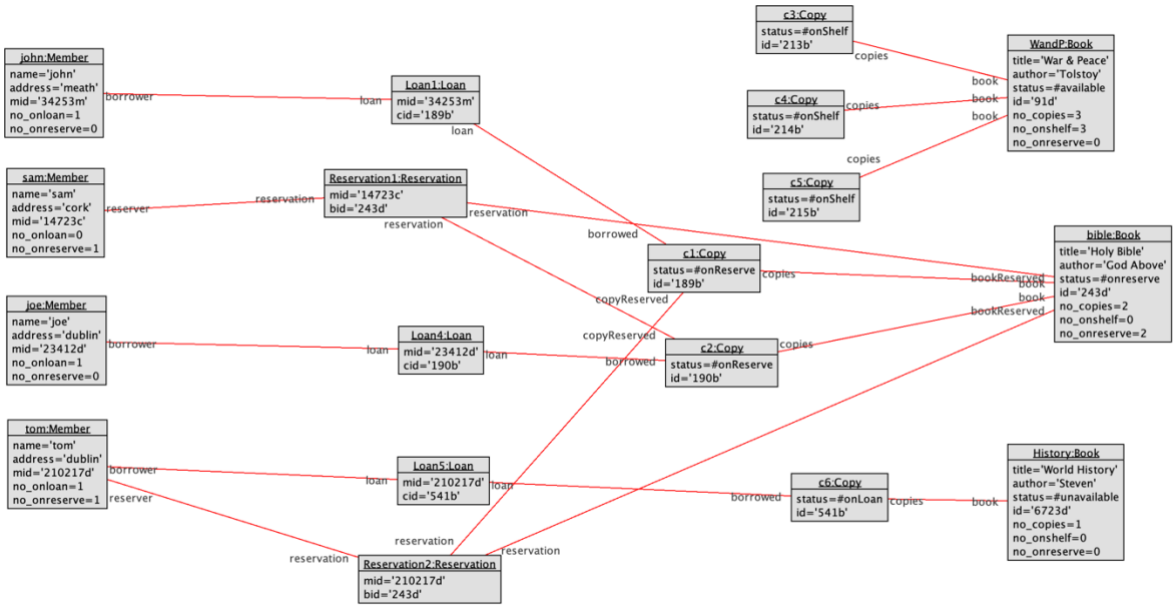




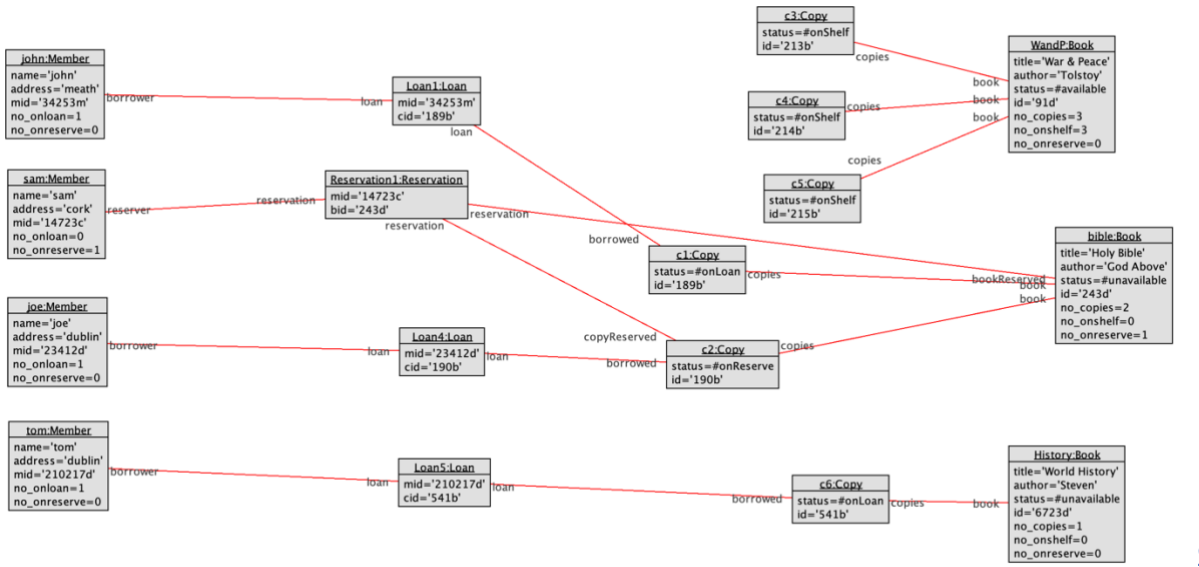


Reserving a book

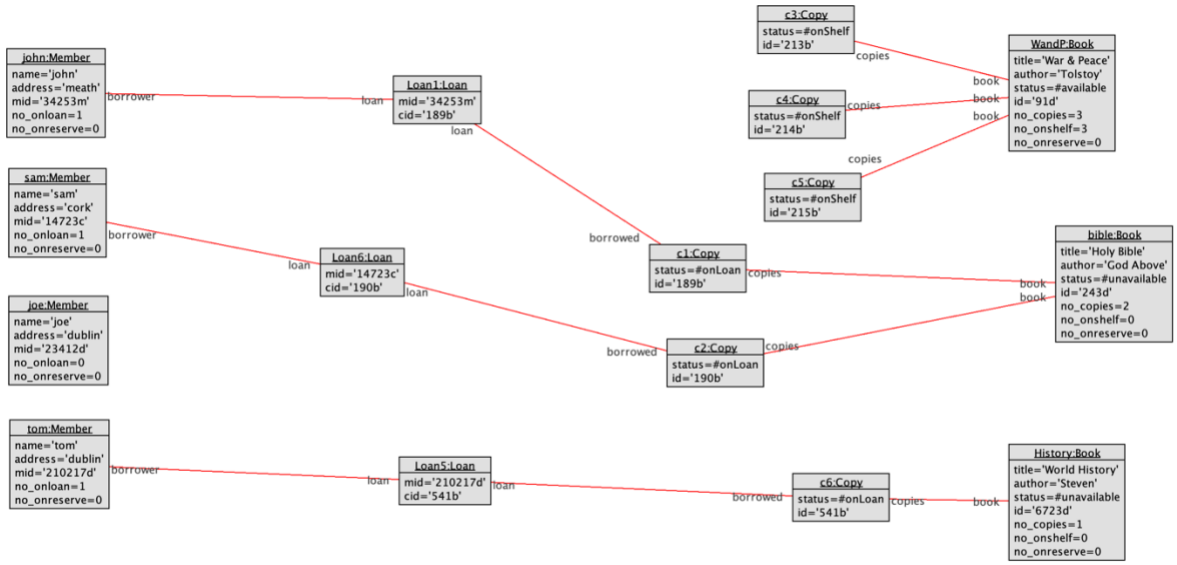




Cancelling a reservation



Borrowing a reserved book



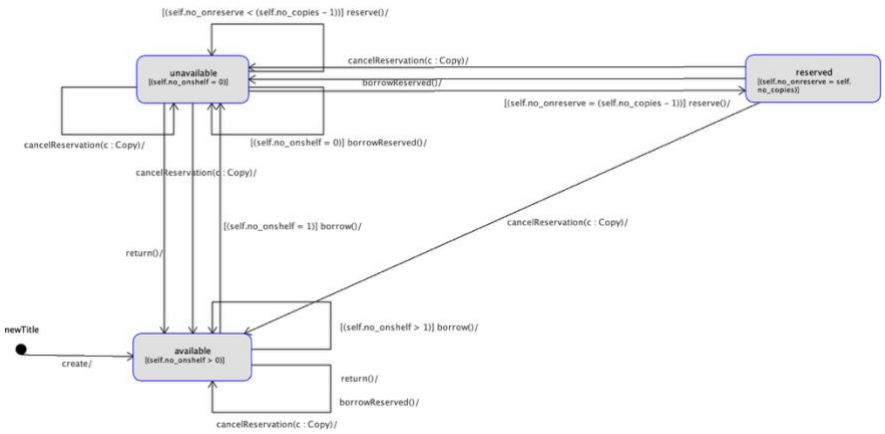
State Machines

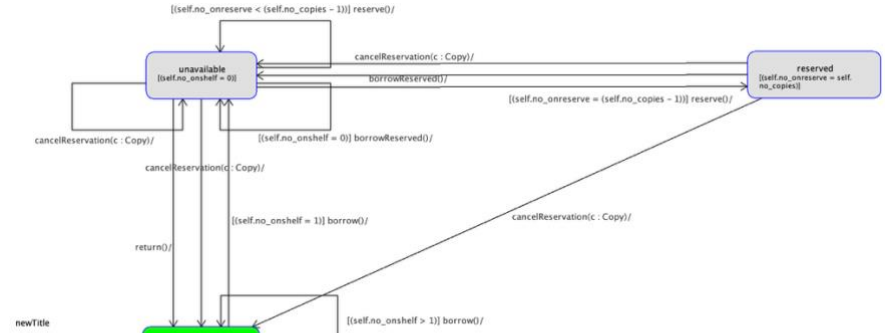
Book Class

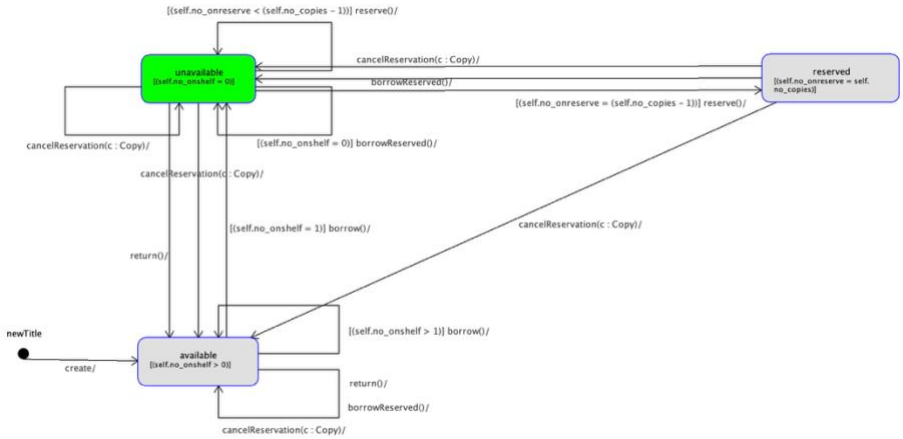
Code

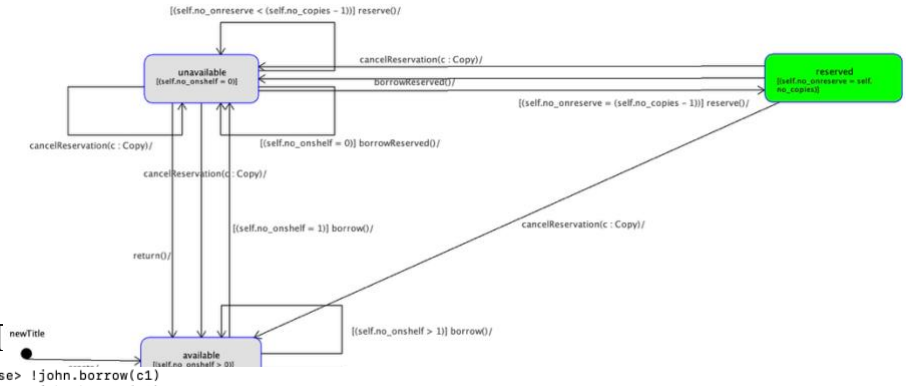
```
statemachines
    psm States
    states
        newTitle : initial
        available       [no_onshelf > 0]
        unavailable     [no_onshelf = 0]
        reserved        [no_onreserve = no_copies]
    transitions
        newTitle -> available  { create }
        available -> unavailable { [no_onshelf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

The different states










```
use> !john.borrow(c1)
use> !john.borrow(c2)
use> !john.return(c2)
use> !john.return(c1)
use> !john.borrow(c1)
use> !sam.borrow(c2)
use> !tom.reserve(bible)
use> !joe.reserve(bible)
use> !john.return(c1)
use> !joe.borrowReserved(Reservation1)
[Error] 2 preconditions in operation call `Member::borrowReserved(self:joe, r:Reservation1)' do not hold:
  memberBorrowResEmpty: r.copyReserved.loan->isEmpty
    r : Reservation = Reservation1
    r.copyReserved : Copy = c2
    r.copyReserved.loan : Loan = Loan4
    r.copyReserved.loan : Set(Loan) = Set{Loan4}
    r.copyReserved.loan->isEmpty : Boolean = false

  memberBorrowResInclusion: self.reservation->includes(r)
    self : Member = joe
    self.reservation : Set(Reservation) = Set{Reservation2}
    r : Reservation = Reservation1
    self.reservation->includes(r) : Boolean = false

  call stack at the time of evaluation:
    1. Member::borrowReserved(self:joe, r:Reservation1) [caller: joe.borrowReserved(Reservation1)@<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::borrowReserved(self:joe, r:Reservation1)'.
use> !joe.borrowReserved(Reservation2)
use> !sam.return(c2)
use> !tom.borrowReserved(Reservation1)
use> !john.reserve(bible)
use> !john.cancelReservation(Reservation3)
use> !john.reserve(bible)
use> !tom.return(c2)
use> !john.cancelReservation(Reservation4)
use> !tom.borrow(c2)
use> !john.reserve(bible)
use> !sam.reserve(bible)
```

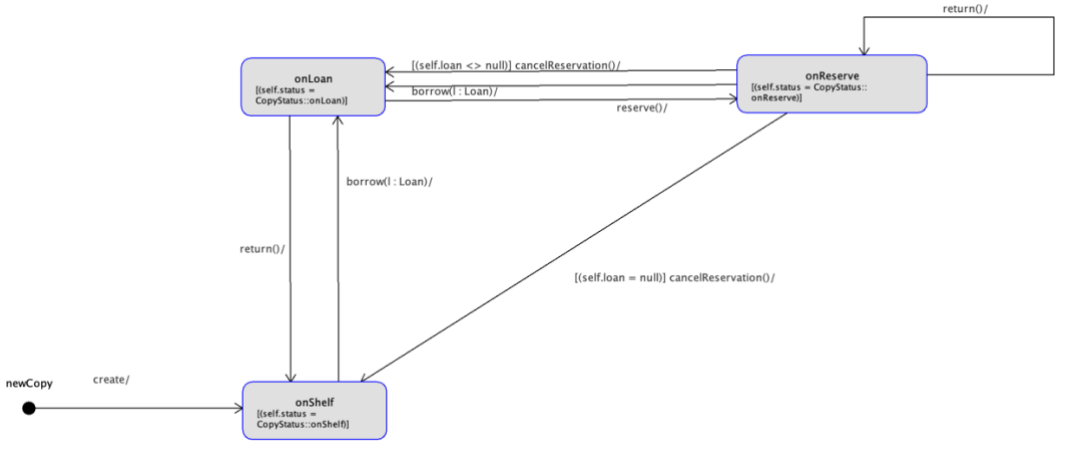
```
use> !sam.cancelReservation(Reservation6)
use> !sam.reserve(bible)
use> !tom.return(c2)
use> !joe.return(c1)
use> !sam.cancelReservation(Reservation7)
use> !john.cancelReservation(Reservation5)
use>
```

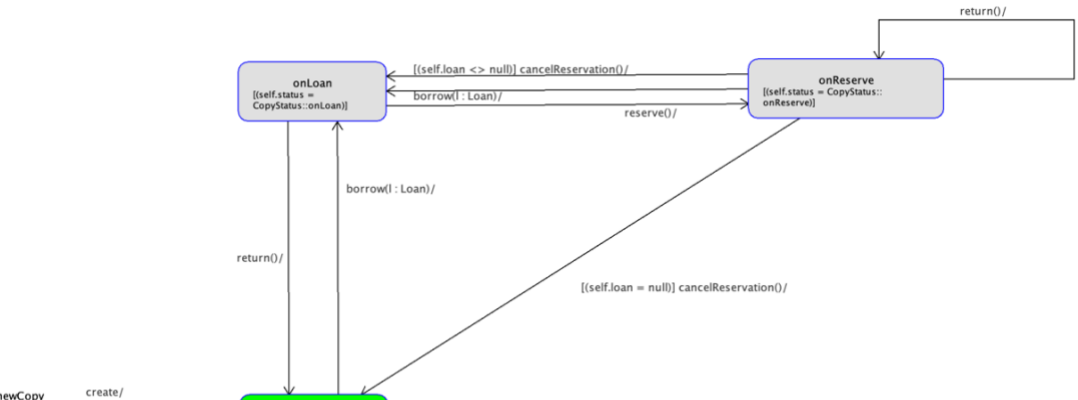
Copy Class

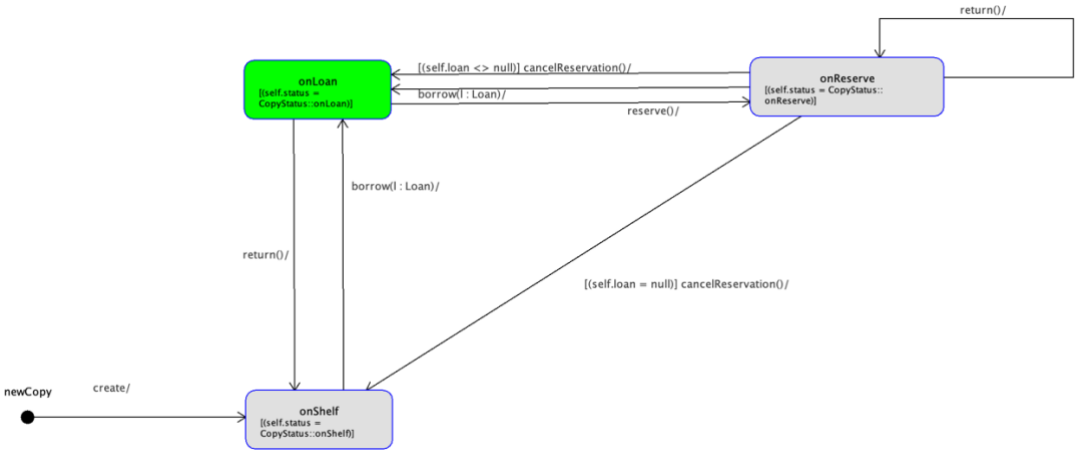
Code

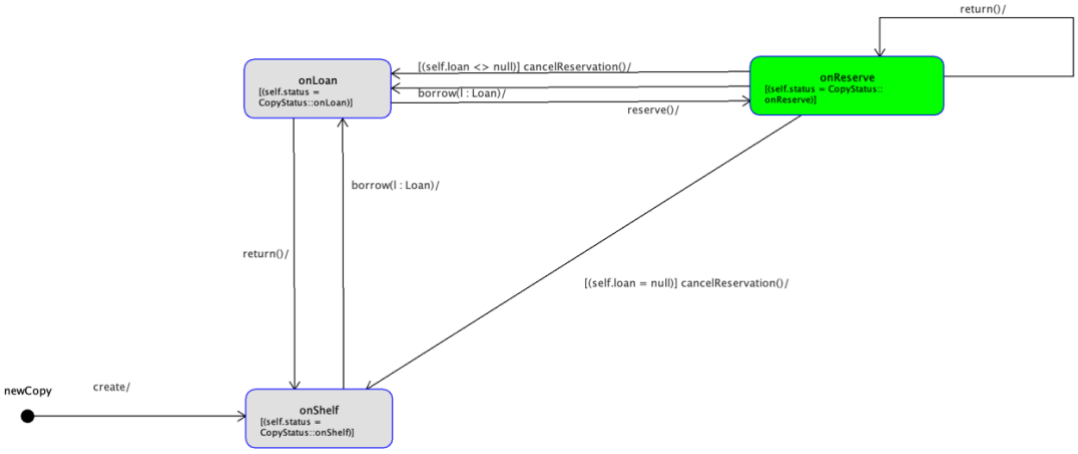
```
statemachines
psm States
states
    newCopy : initial
    onShelf     [status = #onShelf]
    onLoan      [status = #onLoan]
    onReserve   [status = #onReserve]
transitions
    newCopy -> onShelf { create }
    onShelf -> onLoan { borrow() }
    onLoan -> onShelf { return() }
    onLoan -> onReserve { reserve() }
    onReserve -> onShelf { [self.loan = null] cancelReservation() }
    onReserve -> onLoan { [self.loan <> null] cancelReservation() }
    onReserve -> onLoan { borrow() }
    onReserve -> onReserve { return() }
end
```

The different states









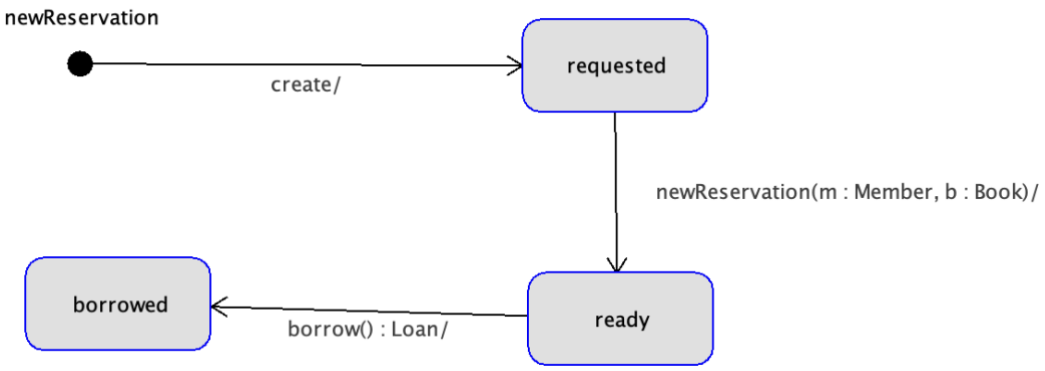

```
use> !john.borrow(c2)
use> !john.return(c2)
use> !john.borrow(c2)
use> !sam.borrow(c1)
use> !tom.reserve(bible)
use> !tom.cancelReservation(Reservation1)
use> !tom.reserve(bible)
use> !joe.reserve(bible)
use> !john.return(c2)
use> !tom.borrowReserved(Reservation2)
use> !john.reserve(bible)
use> !tom.return(c2)
use> !john.cancelReservation(Reservation4)
```

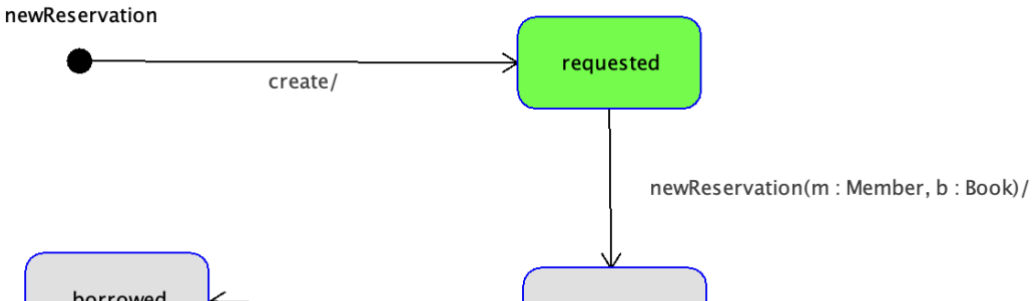
Reservation Class

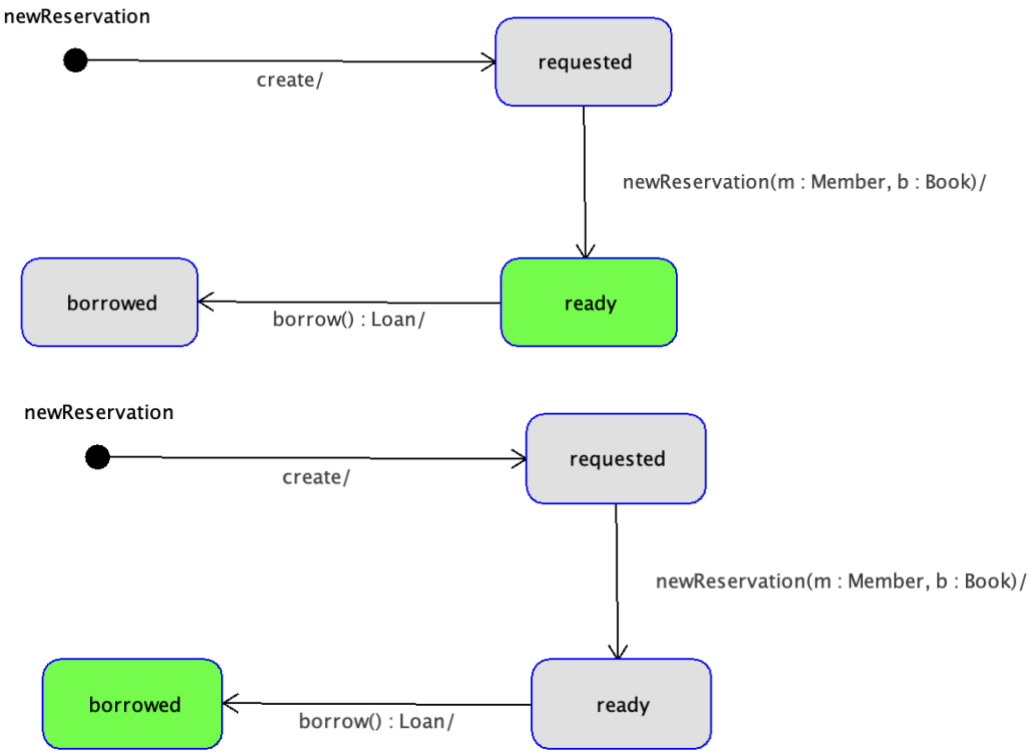
Code

```
statemachines
psm States
states
    newReservation : initial
    requested
    ready
    borrowed
transitions
    newReservation ->  requested  { create }
    requested -> ready { newReservation() }
    ready -> borrowed {borrow()}
end
```

The different states







Test driving

```
use> !new Reservation
use> !tom.borrow(c6)
use> !sam.reserve(History)
use> !tom.return(c6)
use> !sam.borrowReserved(Reservation2)
```

Discussion and Analysis

From my analysis of the diagrams of the system, it seems to me that the system has moderate coupling and moderate cohesion. For coupling,

elements depend on the Loan and Reservation Classes so not directly on each other and for cohesion, most classes are responsible for themselves however they still could both be improved. Some classes have operations and responsibilities that do not align with the responsibilities of the class such as the Member class. It creates the reservation/loan object which does not align with its responsibility. For example, the link between member and Loan/Reservation could be improved by implementing another class between them. It could take the responsibility of creating a Loan or Reservation to increase cohesion and it could also reduce the dependency of member on Loan/Reserved decreasing coupling. Overall, the system is effective but could be improved in the future.

USE Code

```
model Library


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


class Book
    attributes
        title : String
        author : String
        status : BookStatus init = #available
        id: String;
        no_copies : Integer init = 1
        no_onshelf : Integer init = 1
        no_onreserve : Integer init = 0

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

        borrowReserved()
        begin
            self.no_onreserve := self.no_onreserve - 1;
            if(self.no_onshelf = 0) then
```

```
    reserve()
    begin
        self.no_onreserve := self.no_onreserve + 1;
        if(self.no_onreserve = self.no_copies) then
            self.status := #onreserve;
        end
    end

    cancelReservation(c : Copy)
    begin
        if(c.loan <> null) then
            self.no_onreserve := self.no_onreserve - 1;
            self.status := #unavailable;
        else
            self.no_onshelf := self.no_onshelf + 1;
            self.no_onreserve := self.no_onreserve - 1;
            self.status := #available;
        end;
        c.cancelReservation();
    end


  statemachines
    psm States
    states
        newTitle : initial
        available       [no_onshelf > 0]
        unavailable     [no_onshelf = 0]
        reserved        [no_onreserve = no_copies]
    transitions
        newTitle -> available  { create }
        available -> unavailable { [no_onshelf = 1] borrow() }
        available -> available { [no_onshelf > 1] borrow() }
        available -> available { return() }
        available -> available {borrowReserved()}
        available -> available {cancelReservation()}
        unavailable -> unavailable { [no_onreserve < no_copies -1] reserve() }
        unavailable -> reserved { [no_onreserve = no_copies - 1] reserve() }
        unavailable -> unavailable { [no_onshelf = 0] borrowReserved()}
        unavailable -> unavailable {cancelReservation()}
        unavailable -> available {cancelReservation()}
        unavailable -> available { return() }
        reserved -> unavailable { borrowReserved()}
        reserved -> unavailable {cancelReservation()}
        reserved -> available {cancelReservation()}
    end
end

class Copy
  attributes
    status : CopyStatus init = #onShelf
    id : String
  operations
    return()
    begin
        if(self.status <> #onReserve) then
            self.book.return();
            self.status := #onShelf;
        end
    end

    borrow( l : Loan)
    begin
        self.status := #onLoan;
        if (self.reservation = null) then
            self.book.borrow();
        else
            self.book.borrowReserved();
        end
    end

    cancelReservation()
    begin
        if(self.loan <> null) then
            self.status := #onLoan;
        else
            self.status := #onShelf;
        end
    end
```

```
    reserve()
    begin
        self.status := #onReserve;
        self.book.reserve()
    end

    statemachines
    psm States
    states
        newCopy : initial
        onShelf     [status = #onShelf]
        onLoan      [status = #onLoan]
        onReserve   [status = #onReserve]
    transitions
        newCopy -> onShelf { create }
        onShelf -> onLoan { borrow() }
        onLoan -> onShelf { return() }
        onLoan -> onReserve { reserve() }
        onReserve -> onShelf { [self.loan = null] cancelReservation() }
        onReserve -> onLoan { [self.loan <> null] cancelReservation() }
        onReserve -> onLoan { borrow() }
        onReserve -> onReserve { return() }
    end

end

class Member
  attributes
    name : String
    address : String
    mid : String
    no_onloan : Integer init = 0
    no_onreserve : Integer init = 0
  operations
    okToBorrow() : Boolean
    begin
      if (self.no_onloan < 2) then
        result := true
      else
        result := false
      end
    end

    okToReserve() : Boolean
    begin
      if (self.no_onreserve < 2) then
        result := true
      else
        result := false
      end
    end

    borrow(c : Copy)
    begin
        declare ok : Boolean, l : Loan;
        ok := self.okToBorrow();
        if( ok ) then
            l := new Loan;
            l.cid := c.id;
            l.mid := self.mid;
            insert (self, l) into HasBorrowed;
            self.no_onloan := self.no_onloan + 1;
            l.newLoan(self,c);
        end
    end
```

```
    borrowReserved(r:Reservation)
    begin
        declare ok : Boolean, l : Loan;
        ok := self.okToBorrow();
        if( ok ) then
            l := r.borrow();
            destroy r;
            insert(self,l) into HasBorrowed;
            self.no_onloan := self.no_onloan + 1;
            self.no_onreserve := self.no_onreserve - 1;
        end
    end

    reserve(b : Book)
    begin
      declare ok: Boolean, r : Reservation;
      ok := self.okToReserve();
      if( ok ) then
          r := new Reservation;
          r.bid := b.id;
          r.mid := self.mid;
          insert (self, r) into HasReserved;
          self.no_onreserve := self.no_onreserve + 1;
          r.newReservation(self,b)
      end
    end


    return(c : Copy)
    begin
        self.no_onloan := self.no_onloan - 1;
        destroy c.loan;
        c.return();
    end

    cancelReservation(r : Reservation)
    begin
        r.bookReserved.cancelReservation(r.copyReserved);
        self.no_onreserve := self.no_onreserve - 1;
        destroy r;
    end
end

class Loan
  attributes
    mid : String
    cid : String
  operations
    newLoanReserved(m : Member, c : Copy)
    begin
        insert(self,c) into CopyBorrowed;
        c.borrow(self);
    end

    newLoan(m : Member, c : Copy)
    begin
        insert(self,c) into CopyBorrowed;
        c.borrow(self);
    end

end

class Reservation
  attributes
    mid : String
    bid : String
  operations
    borrow() : Loan
    begin
        declare l : Loan;
        l := new Loan;
        l.cid := self.copyReserved.id;
        l.mid := self.mid;
        l.newLoanReserved(self.reserver,self.copyReserved);
        result := l;
    end
```

```
    newReservation(m : Member, b: Book)
    begin
        declare c: Copy;
        insert(self,b) into BookReserved;
        for x in b.copies do
            if(x.status = #onLoan) then
                c := x
            end
        end;
        c.reserve();
        insert(self,c) into CopyReserved;
    end

    statemachines
    psm States
    states
        newReservation : initial
        requested
        ready
        borrowed
    transitions
        newReservation ->  requested  { create }
        requested -> ready { newReservation() }
        ready -> borrowed {borrow()}
    end
end


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

association CopyBorrowed between
    Loan[0..1]
    Copy[1] role borrowed
end

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

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

association CopyReserved between
    Reservation[0..1]
    Copy[0..1] role copyReserved
end

association BookReserved between
    Reservation[0..*]
    Book[1] role bookReserved
end



constraints


context Book::return()
    post bookReturnIncrease: no_onshelf = no_onshelf@pre + 1

context Member::borrow(c:Copy)
    pre memberBorrowLimit: self.no_onloan < 2
    pre memberStatusCheck: c.status = #onShelf
    pre memberBorrowExclusion: self.loan.borrowed->excludes(c)
    post memberBorrowInclusion: self.loan.borrowed->includes(c)

context Loan::newLoanReserved(m : Member, c : Copy)
    pre loanNewLoanResLimit: m.no_onloan < 2
    pre loanNewLoanResExclusion: self.borrowed->excludes(c)
    post loanNewLoanResInclusion: self.borrowed->includes(c)
```

```
context Member:: return(c:Copy)
    pre memberReturnExclusion: self.loan.borrowed->includes(c)
    post memberReturnInclusion: self.loan.borrowed->excludes(c)

context Member:: reserve(b : Book)
    pre memberReserveMinimum: b.no_onshelf = 0
    post memberReserveInclusion: self.reservation.bookReserved->includes(b)

context Member:: borrowReserved(r:Reservation)
    pre memberBorrowResEmpty: r.copyReserved.loan -> isEmpty()
    pre memberBorrowResStatus: r.copyReserved.status = #onReserve
    pre memberBorrowResInclusion: self.reservation -> includes(r)
    post memberBorrowResExclusion: self.reservation -> excludes(r)

context Member:: cancelReservation(r : Reservation)
    pre memberCancelResInclusion: self.reservation -> includes(r)
    post memberCancelResExclusion: self.reservation -> excludes(r)



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

context Member
    inv loanMin: self.no_onloan >= 0
    inv reserveMin: self.no_onreserve >= 0

context Book
    inv numBookCopies: self.no_copies >= 0
    inv numBookShelf: self.no_onshelf >= 0
    inv numBookReserve: self.no_onreserve >= 0
    inv BookStatusInv: self.status = #available or self.status = #unavailable or self.status = #onreserve
```