Software Engineering 2 Assignment – Report

C23380123 Ciaran Duggan

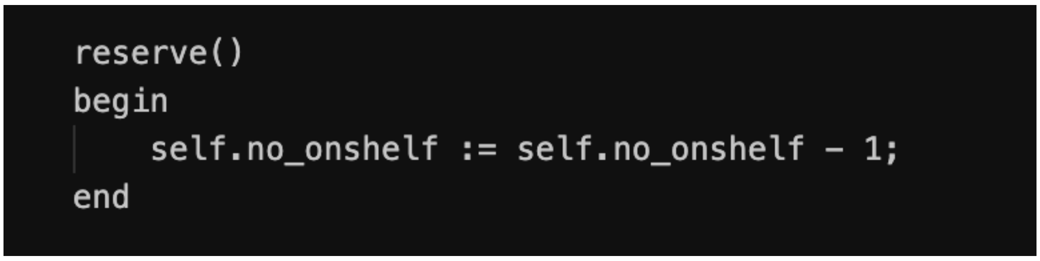
Introduction:
I chose to extend and test a more comprehensive USE model for the library system in USE for my assignment. I extended the USE model of the library system from the Software Engineering 1 module in semester one. This includes the new use cases, preconditions, post conditions, invariants and constraints. It also has the state machine along with diagrams and both the use and soil file have also been submitted.

Firstly, I added two new use cases that are not borrow () or return () into the library. The two that I added were:

payFine ():
This operation allows the library member to pay their fine if they have one. The member will have received a fine if they did not return a book or if they returned it late.
Scenario:
The user has a fine to pay.
User requests to pay their fine.
The user pays the fine.
The fine is reset to 0 and removed on the member's account.

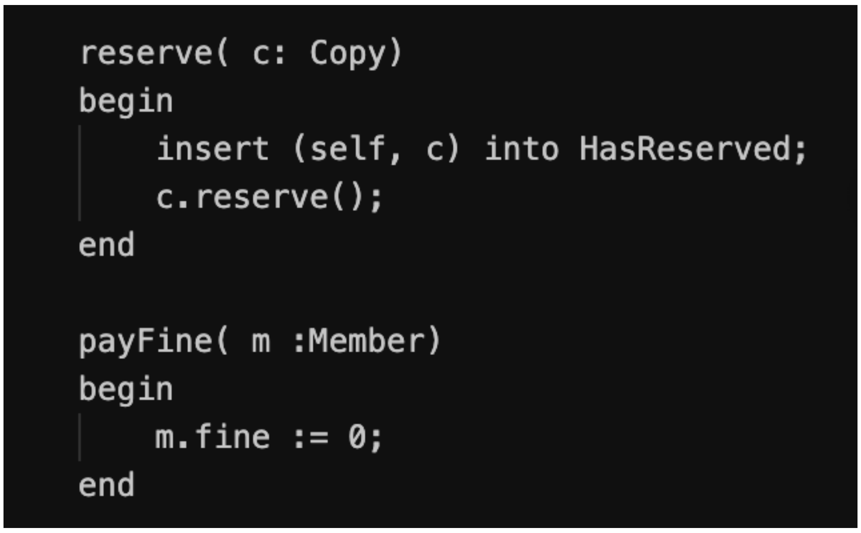
Reserve ():
The reserve use case allows the user to reserve a book. This book will then be put aside for this member. No one else will be able to borrow the book until this user borrows it. This is common in many libraries.
Scenario:
User makes a request to reserve a book.
System checks if the user is able to reserve.
The book is added to the member's reserved list.
The copy of the book is added to the library's reserved list.

Here is the code implementation:
Book Class:



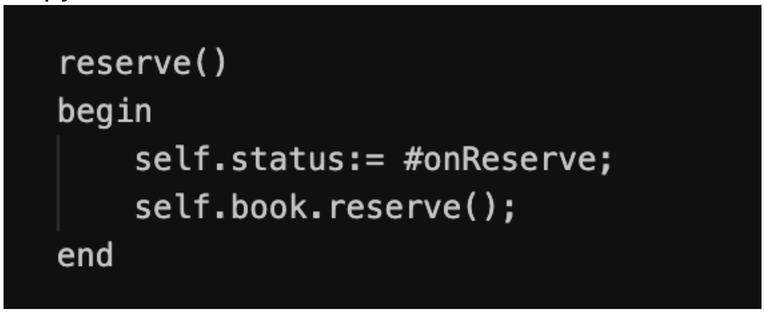
```
reserve()
begin
    self.no_onshelf := self.no_onshelf - 1;
end
```

Member Class:



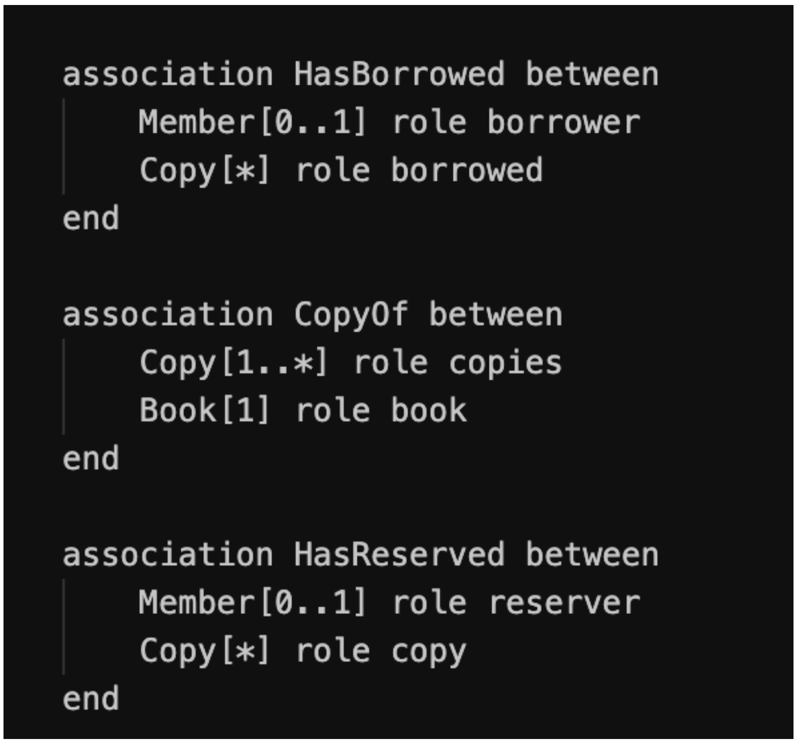
```
reserve( c: Copy)
begin
    insert (self, c) into HasReserved;
    c.reserve();
end

payFine( m :Member)
begin
    m.fine := 0;
end
```

Copy Class:



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

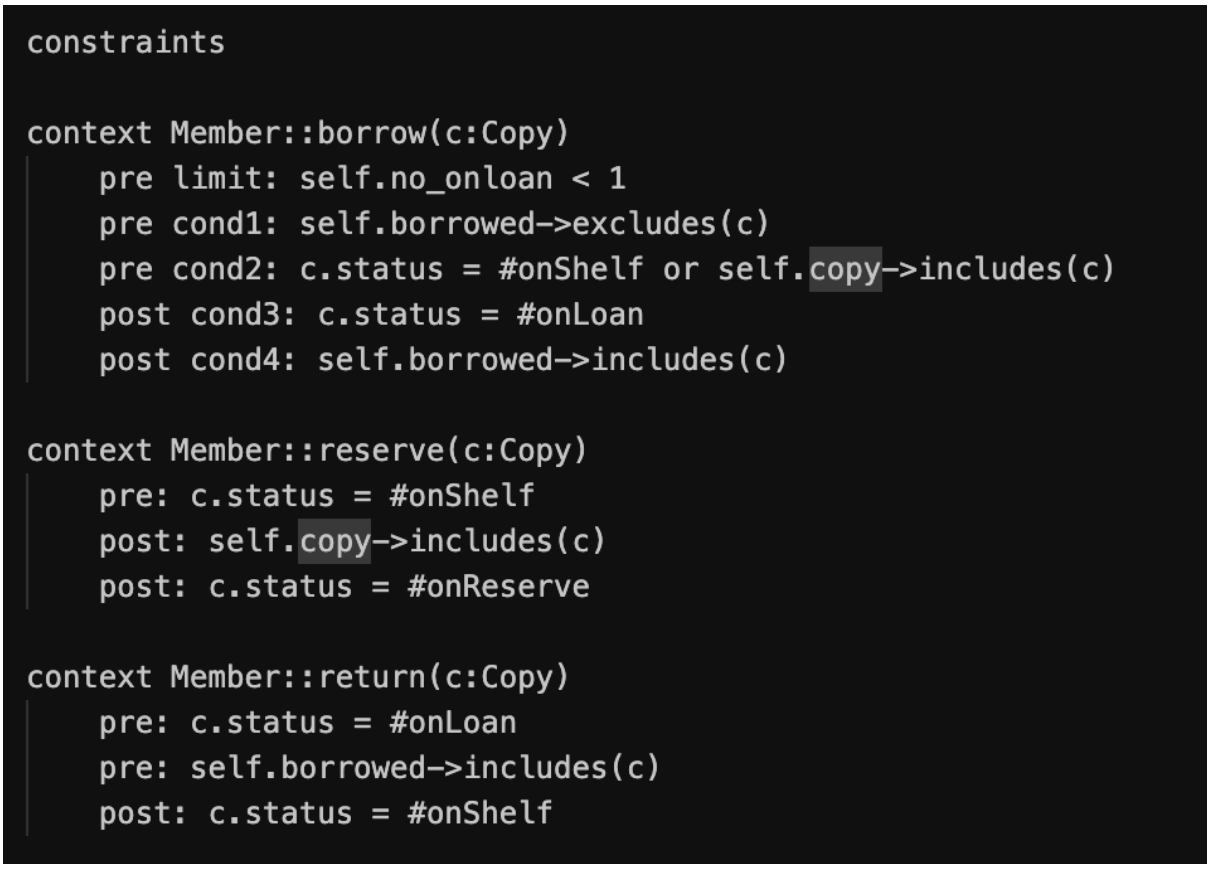
Here are the associations added:



```
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 HasReserved between
    Member[0..1] role reserver
    Copy[*] role copy
end
```

The next part was adding in preconditions, postconditions and invariants. I added the following:

Constraints:



```
constraints

context Member::borrow(c:Copy)
    pre limit: self.no_onloan < 1
    pre cond1: self.borrowed->excludes(c)
    pre cond2: c.status = #onShelf or self.copy->includes(c)
    post cond3: c.status = #onLoan
    post cond4: self.borrowed->includes(c)

context Member::reserve(c:Copy)
    pre: c.status = #onShelf
    post: self.copy->includes(c)
    post: c.status = #onReserve

context Member::return(c:Copy)
    pre: c.status = #onLoan
    pre: self.borrowed->includes(c)
    post: c.status = #onShelf
```

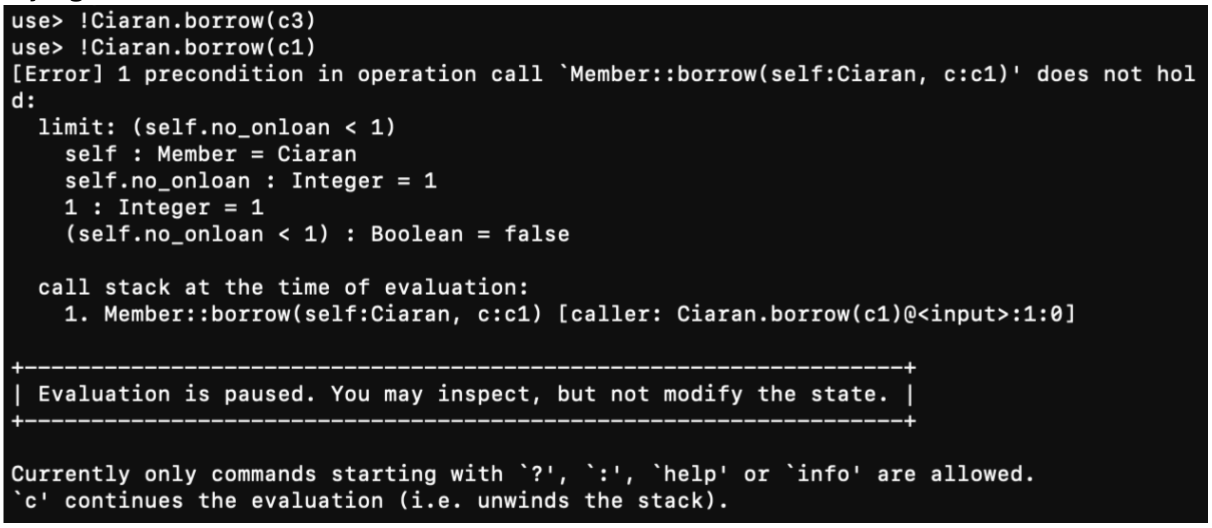
Member borrow() operation:
Preconditions: The member cannot borrow a book if they already are borrowing a book. The book must be either on the shelf or borrowed by the user already (e.g. extending the borrow period) in order to be borrowed.
Post-conditions: The status of the copy must be "onLoan" and the copy must be borrowed.

Member reserve() operation:
Precondition: In order to reserve a copy, the copy must be on the shelf.
Post-condition: The copy's status must be set to "reserved".

Member return() operation:
Precondition: In order to return a copy, it must be on loan already.
Post-condition: The copy's status must be set to "onShelf".

TESTING CONSTRAINTS

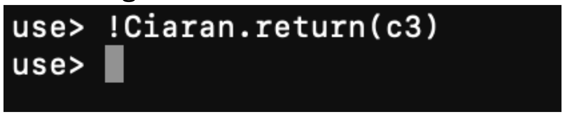
Trying to borrow a second book



```
use> !Ciaran.borrow(c3)
use> !Ciaran.borrow(c1)
[Error] 1 precondition in operation call `Member::borrow(self:Ciaran, c:c1)' does not hol
d:
  limit: (self.no_onloan < 1)
    self : Member = Ciaran
    self.no_onloan : Integer = 1
    1 : Integer = 1
    (self.no_onloan < 1) : Boolean = false

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

Success the user can only borrow one book at a time

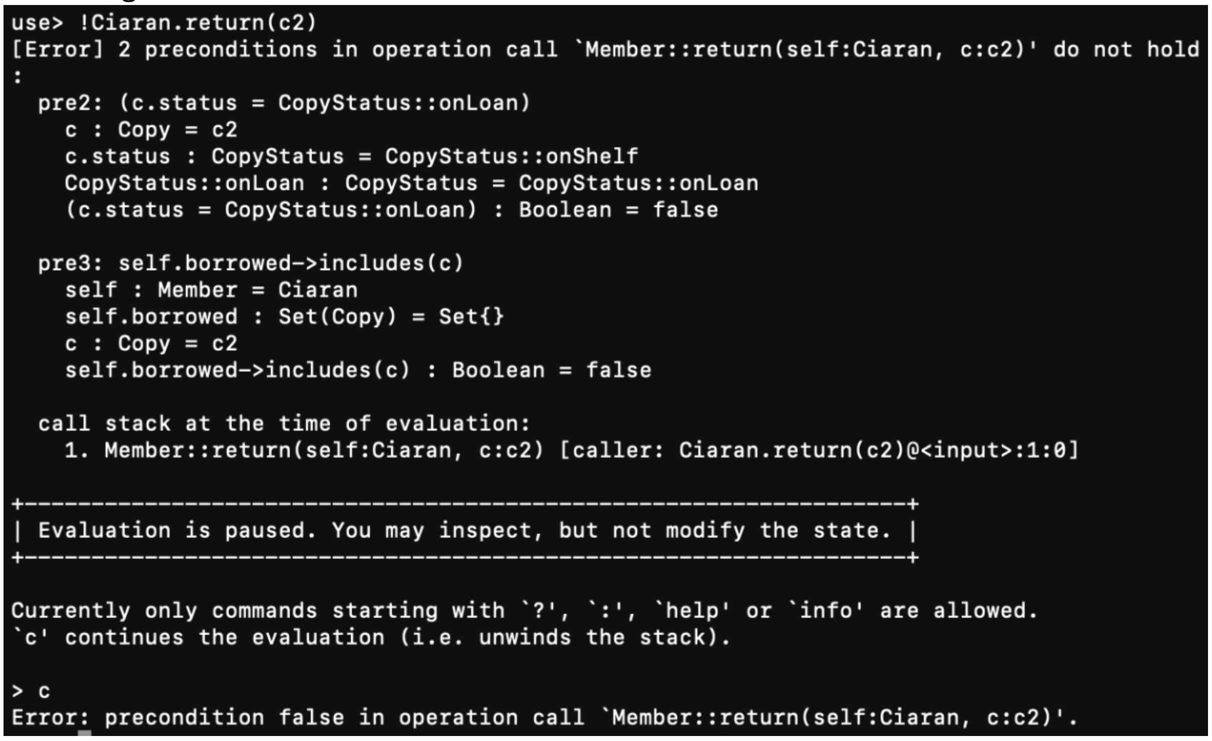
Returning a book.



```
use> !Ciaran.return(c3)
use> 
```

Success book returned.

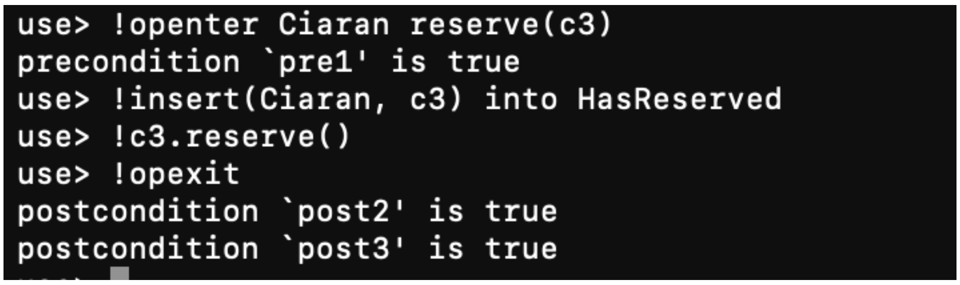
Returning a book that was not borrowed.



```
use> !Ciaran.return(c2)
[Error] 2 preconditions in operation call `Member::return(self:Ciaran, c:c2)' do not hold
:
  pre2: (c.status = CopyStatus::onLoan)
    c : Copy = c2
    c.status : CopyStatus = CopyStatus::onShelf
    CopyStatus::onLoan : CopyStatus = CopyStatus::onLoan
    (c.status = CopyStatus::onLoan) : Boolean = false

  pre3: self.borrowed->includes(c)
    self : Member = Ciaran
    self.borrowed : Set(Copy) = Set{}
    c : Copy = c2
    self.borrowed->includes(c) : Boolean = false

  call stack at the time of evaluation:
    1. Member::return(self:Ciaran, c:c2) [caller: Ciaran.return(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::return(self:Ciaran, c:c2)'.
```

Success. Cannot be done.

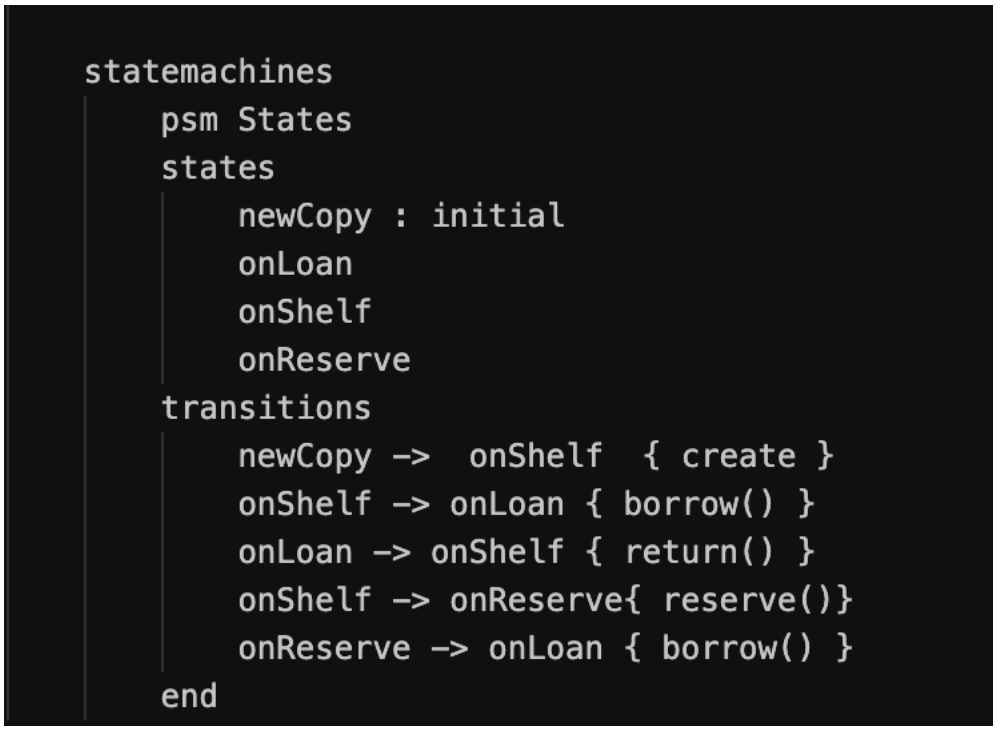
OPENTER AND OPEXIT



```
use> !openter Ciaran reserve(c3)
precondition `pre1' is true
use> !insert(Ciaran, c3) into HasReserved
use> !c3.reserve()
use> !opexit
postcondition `post2' is true
postcondition `post3' is true
```

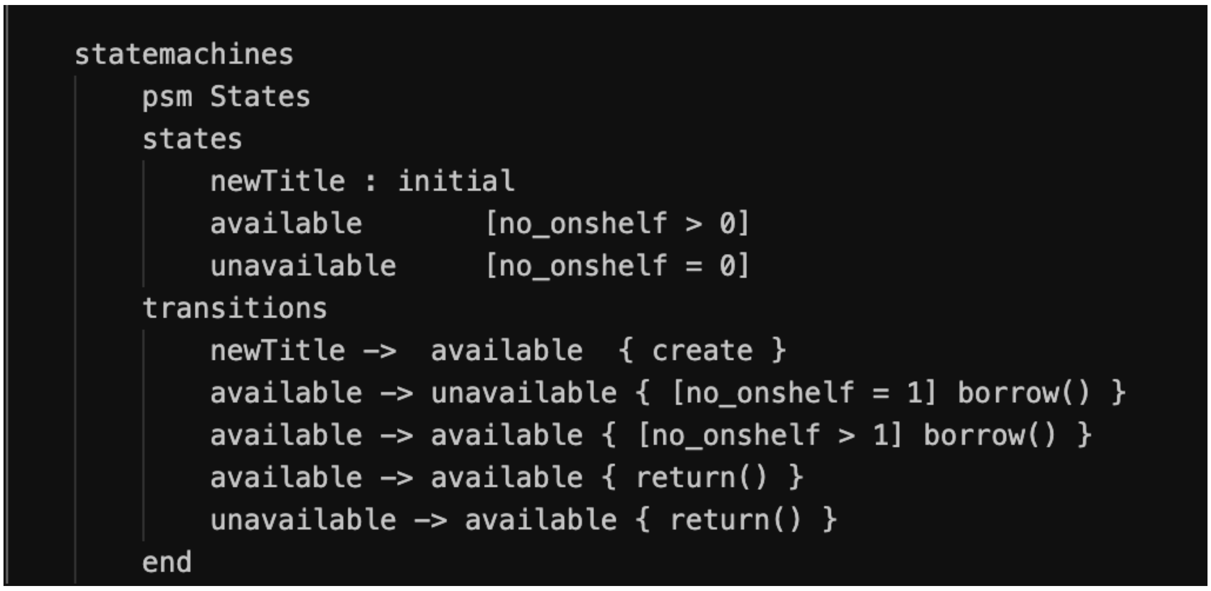
STATE MACHINE

COPY CLASS:



```
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() }
    end
```

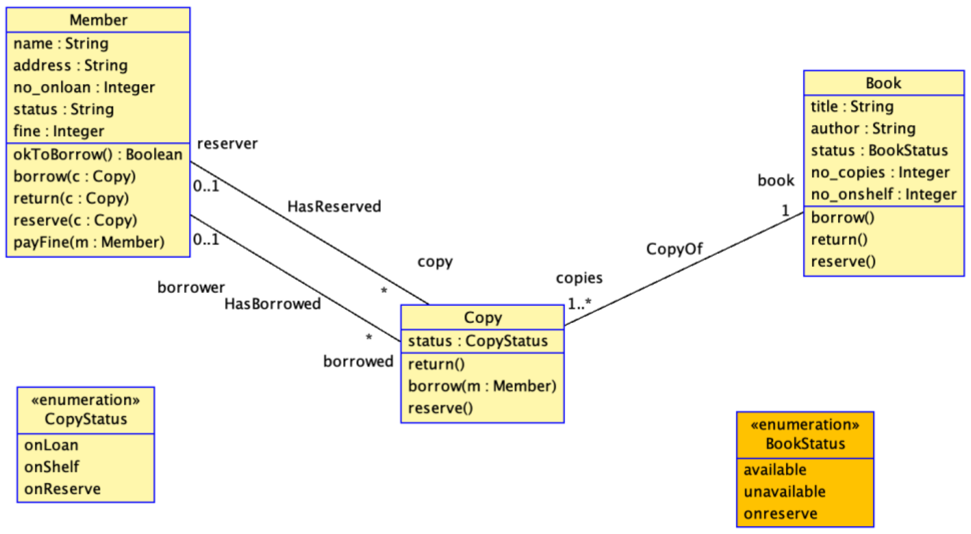
BOOK CLASS:



```
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
```

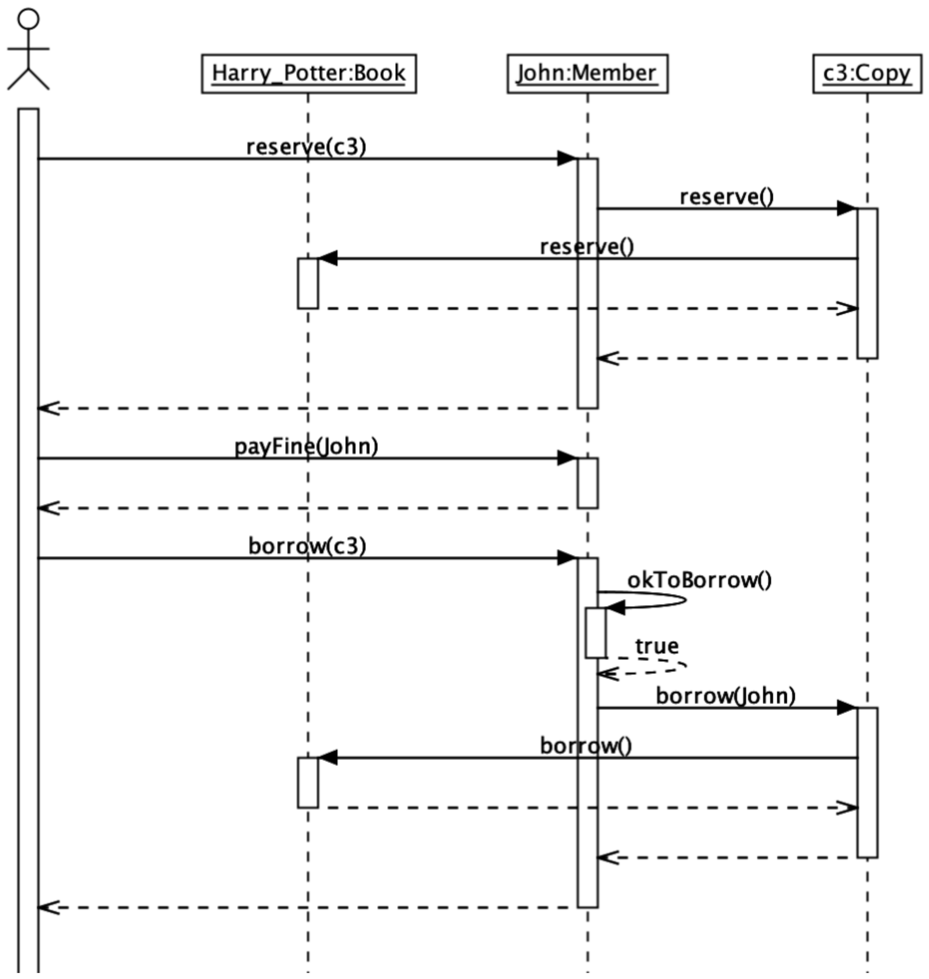
DIAGRAMS

Class diagram

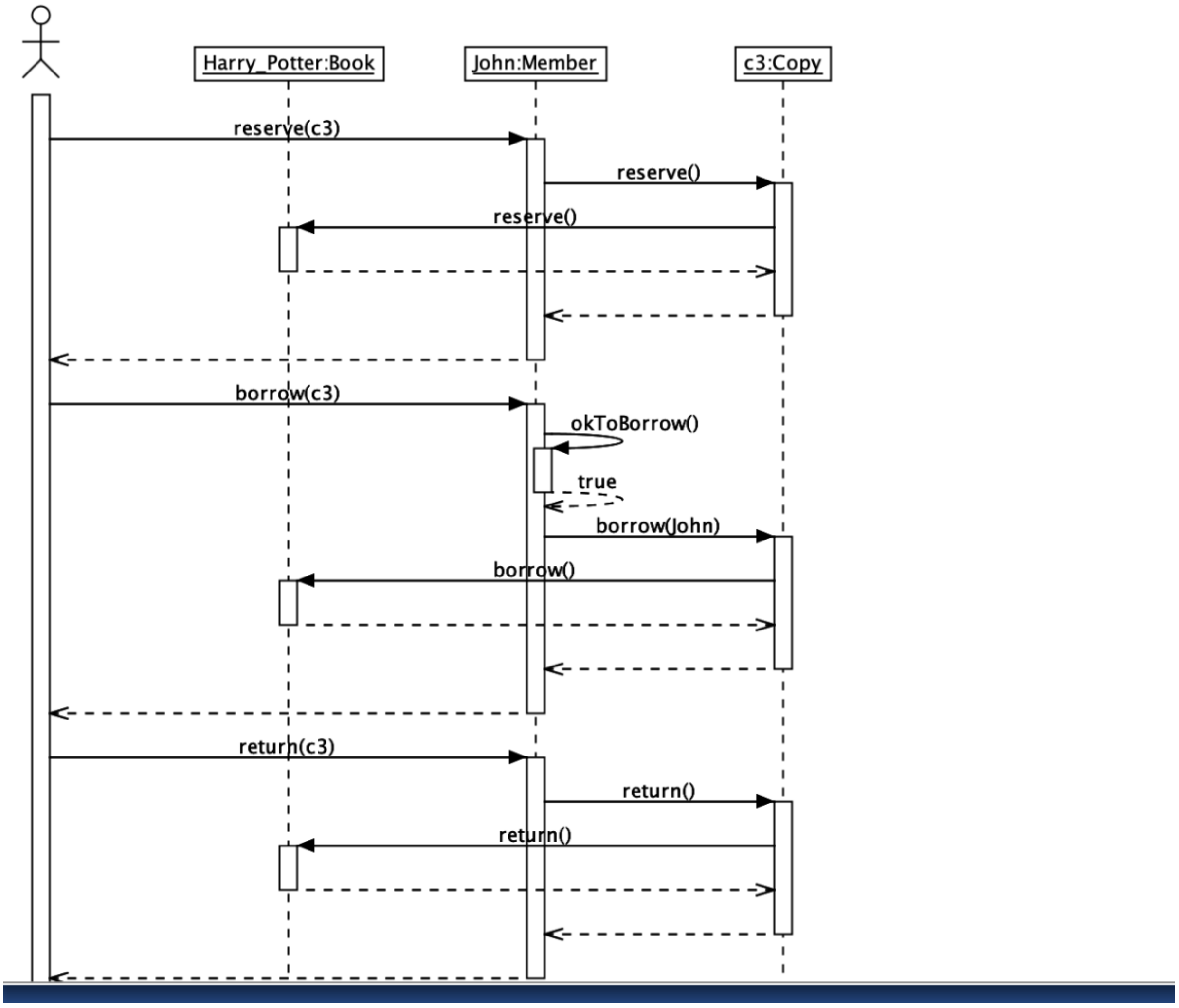


Sequence Diagram

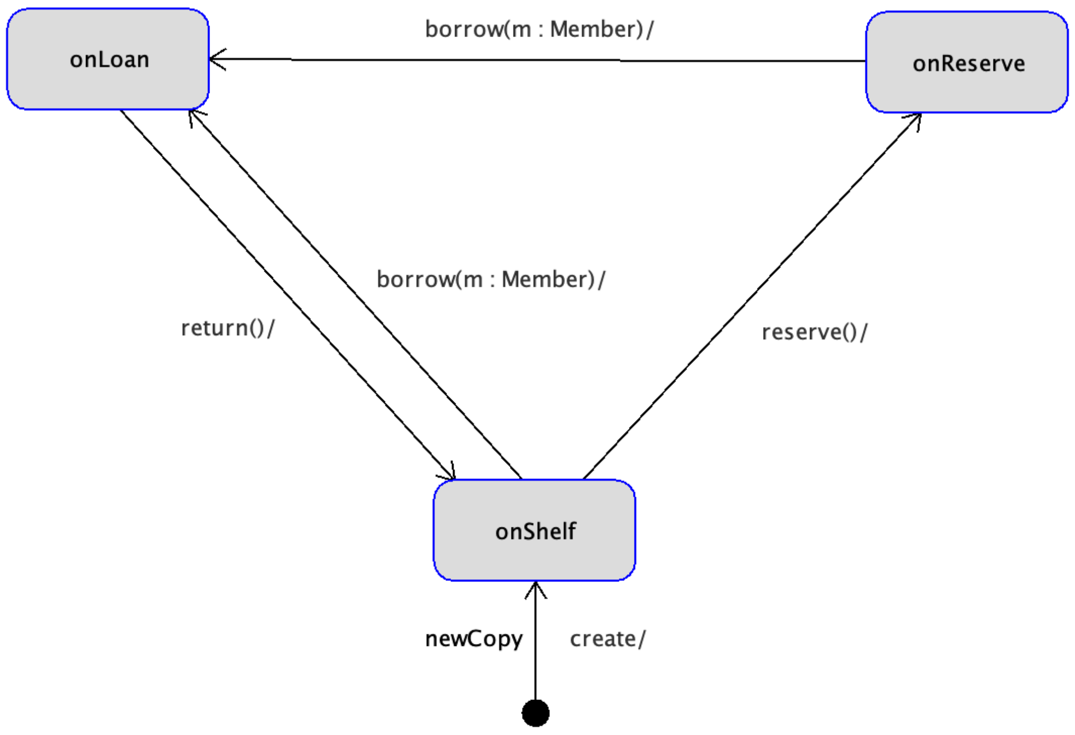
Reserving a copy, paying fine and borrowing a copy.



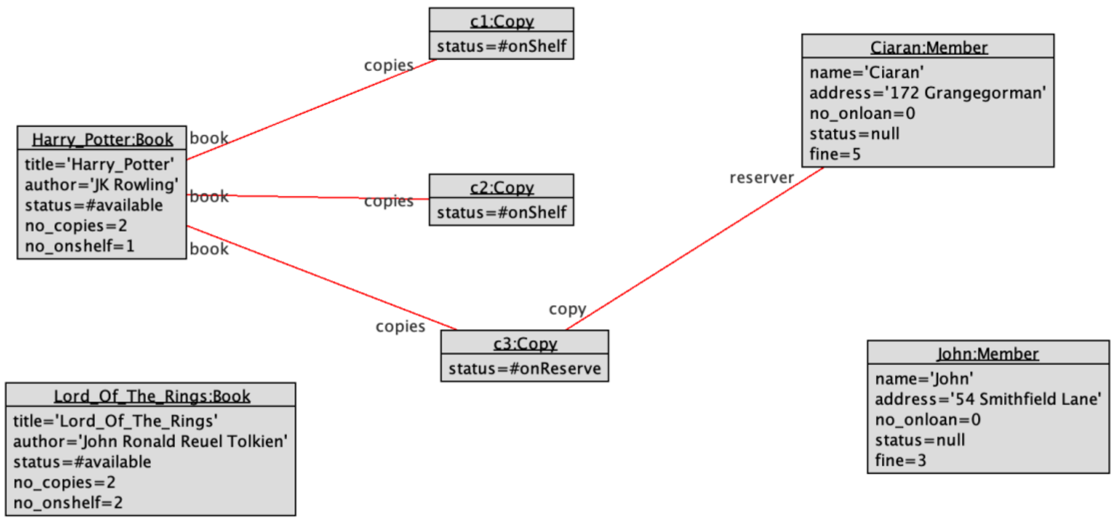
Reserve, borrow and return.



State machine diagram for copy



Object diagram after Openter and Opexit



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
  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

  reserve()
  begin
    self.no_onshelf := self.no_onshelf - 1;
  end

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

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

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

    reserve( c: Copy)
    begin
      insert (self, c) into HasReserved;
      c.reserve();
    end

    payFine( m :Member)
    begin
      m.fine := 0;
    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 HasReserved between
   Member[0..1] role reserver
   Copy[*] role copy
end

constraints

context Member::borrow(c:Copy)
   pre limit: self.no_onloan < 1
   pre cond1: self.borrowed->excludes(c)
   pre cond2: c.status = #onShelf or self.copy->includes(c)
   post cond3: c.status = #onLoan
   post cond4: self.borrowed->includes(c)

context Member::reserve(c:Copy)
   pre: c.status = #onShelf
   post: self.copy->includes(c)
   post: c.status = #onReserve

context Member::return(c:Copy)
   pre: c.status = #onLoan
   pre: self.borrowed->includes(c)
   post: c.status = #onShelf
```

SOIL CODE

```
!new Member('Ciaran')
!Ciaran.name := 'Ciaran'
!Ciaran.no_onloan := 0
!Ciaran.address := '172 Grangegorman'
!Ciaran.fine := 5

!new Book('Harry_Potter')
!Harry_Potter.title := 'Harry_Potter'
!Harry_Potter.author := 'JK Rowling'
!Harry_Potter.no_copies := 2
!Harry_Potter.no_onshelf := 2

!new Copy('c1')
```

```
!c1.status := #onShelf
!insert(c1, Harry_Potter) into CopyOf

!new Copy('c2')
!c2.status := #onShelf
!insert (c2,Harry_Potter) into CopyOf

!new Member('John')
!John.name := 'John'
!John.no_onloan := 0
!John.address := '54 Smithfield Lane'
!John.fine := 3

!new Copy('c3')
!c3.status := #onShelf
!insert(c3,Harry_Potter) into CopyOf

!new Book('Lord_Of_The_Rings')
!Lord_Of_The_Rings.title := 'Lord_Of_The_Rings'
!Lord_Of_The_Rings.author := 'John Ronald Reuel Tolkien'
```