**Software Engineering 2 Assignment**

Extension of the Library specification example

# Due Date: 25/04/2018

# Date Completed: 03/04/2018

**Completed by:**

# **William Carey**

***C16315253***

# **DT228/2**

Table of Contents

[Lib1.use: 3](#_Toc510530455)

[Class Diagram 12](#_Toc510530456)

[Sequence Diagrams 13](#_Toc510530457)

[Object Diagrams 23](#_Toc510530459)

[State Machine for Class Book 24](#_Toc510530460)

[Class Invariant: 25](#_Toc510530461)

[Testing Constraints on the Command Prompt 26](#_Toc510530462)

[Report on the Library Specification 31](#_Toc510530463)

# Lib1.use:

model Library

--All the books associated with the system

enum BookStatus { available, unavailable, onreserve}

class Book

attributes

title : String

author : String

status : BookStatus init = #available

ShortTerm : Boolean init = false

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

pre copiesOnShelf: no\_copies >0

post: no\_onshelf = no\_onshelf@pre - 1

return()

begin

self.no\_onshelf := self.no\_onshelf + 1;

self.status := #available

end

pre copiesOnShelf: no\_copies >= 0

post: no\_onshelf = no\_onshelf@pre + 1

newCopy(c : Copy)

begin

insert (c,self) into CopyOf;

c.status := 'onShelf';

self.no\_copies := self.no\_copies + 1;

self.no\_onshelf := self.no\_onshelf + 1;

c.ShortTerm := self.ShortTerm

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 Journal

attributes

title : String

author : String

onShelf : Boolean init = true

operations

borrow()

begin

self.onShelf := false

end

pre no\_onshelfTrue: onShelf = true

post: onShelf = false

return()

begin

self.onShelf := true

end

pre no\_onshelfFalse: onShelf = false

post: onShelf = true

end

class Copy

attributes

status : String

ReservedBy : String init = 'NA'

ShortTerm : Boolean

weeksDesired: Integer init = 0

operations

borrow()

begin

self.status := 'onLoan';

self.book.borrow()

end

pre cond1: if ShortTerm = true then weeksDesired <= 1 else weeksDesired <= 3 endif

pre cond2 : weeksDesired > 0

return()

begin

self.status := 'onShelf';

self.book.return()

end

end

--All the people associated with the system

class Member

attributes

name : String

address : String

no\_onloan : Integer init = 0

status : String init = 'Active'

fine : Integer init = 0

operations

borrow(c : Copy)

begin

insert (self, c) into HasBorrowed;

c.ReservedBy := 'NA';

self.no\_onloan := self.no\_onloan + 1;

c.borrow()

end

return(c : Copy)

begin

delete (self, c) from HasBorrowed;

self.no\_onloan := self.no\_onloan - 1;

c.return()

end

--Testing this function in !openter !opexit

Reserve(c : Copy)

--begin

--c.status := 'Reserved';

--c.ReservedBy := self.name;

--end

CancelReserve(c : Copy)

begin

c.status := 'onShelf';

c.ReservedBy := 'NA'

end

end

class Staff

attributes

name : String

address : String

no\_onloan : Integer init = 0

fine : Integer init = 0

operations

--journal work

JournalBorrow(j : Journal)

begin

insert (self, j) into JournalBorrowed;

self.no\_onloan := self.no\_onloan + 1;

j.borrow()

end

JournalReturn(j : Journal)

begin

delete (self, j) from JournalBorrowed;

self.no\_onloan := self.no\_onloan - 1;

j.return()

end

--book work

CopyBorrow(c : Copy)

begin

insert (self, c) into CopyBorrowed;

self.no\_onloan := self.no\_onloan + 1;

c.borrow()

end

CopyReturn(c : Copy)

begin

delete (self, c) from CopyBorrowed;

self.no\_onloan := self.no\_onloan - 1;

c.return()

end

end

--associations

association HasBorrowed between

Member[0..1] role MemberBorrower

Copy[\*] role borrowed

end

association CopyBorrowed between

Staff[0..1] role borrower

Copy[\*] role copyBorrowed

end

association JournalBorrowed between

Staff[0..1] role StaffBorrower

Journal[1] role journalBorrowed

end

association CopyOf between

Copy[0..\*] role copies

Book[1] role book

end

--constraints in the programme

constraints

--All members functions

context Member::borrow(c:Copy)

pre limit: self.no\_onloan < 6

pre status: c.status <> 'onLoan'

pre Resereved:if c.ReservedBy <> 'NA' then c.ReservedBy = self.name else c.ReservedBy = 'NA' endif

pre cond1: self.borrowed->excludes(c)

post cond2: self.borrowed->includes(c)

post status2: c.status = 'onLoan'

context Member::return(c:Copy)

pre status: c.status = 'onLoan'

pre cond1: self.borrowed->includes(c)

post cond2: self.borrowed->excludes(c)

post status2: c.status = 'onShelf'

context Member::Reserve(c : Copy)

pre status: c.status = 'onShelf'

post status2 : c.status = 'Reserved'

context Member::CancelReserve(c:Copy)

pre status: c.status = 'Reserved'

post status2 : c.status = 'onShelf'

--All staff functions

context Staff::JournalBorrow(j : Journal)

pre limit: self.no\_onloan < 12

pre status1: j.onShelf = true

pre cond1: self.journalBorrowed->excludes(j)

post cond2: self.journalBorrowed->includes(j)

post status2: j.onShelf= false

context Staff::JournalReturn(j : Journal)

pre status1: j.onShelf = false

pre cond1: self.journalBorrowed->includes(j)

post cond2: self.journalBorrowed->excludes(j)

post status2: j.onShelf= true

context Staff::CopyBorrow(c:Copy)

pre limit: self.no\_onloan < 12

pre status1: c.status = 'onShelf'

pre cond1: self.copyBorrowed->excludes(c)

post cond2: self.copyBorrowed->includes(c)

post status2: c.status = 'onLoan'

context Staff::CopyReturn(c:Copy)

pre status: c.status = 'onLoan'

pre cond1: self.copyBorrowed->includes(c)

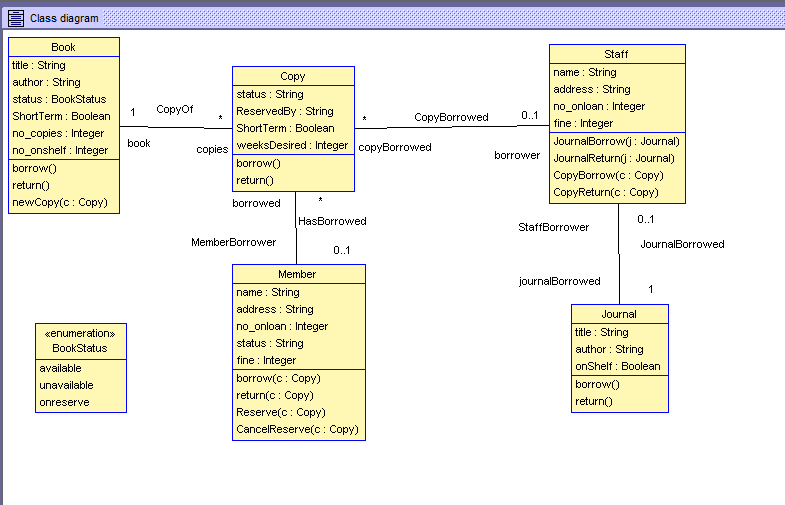
post cond2: self.copyBorrowed->excludes(c)

post status2: c.status = 'onShelf'

context Copy

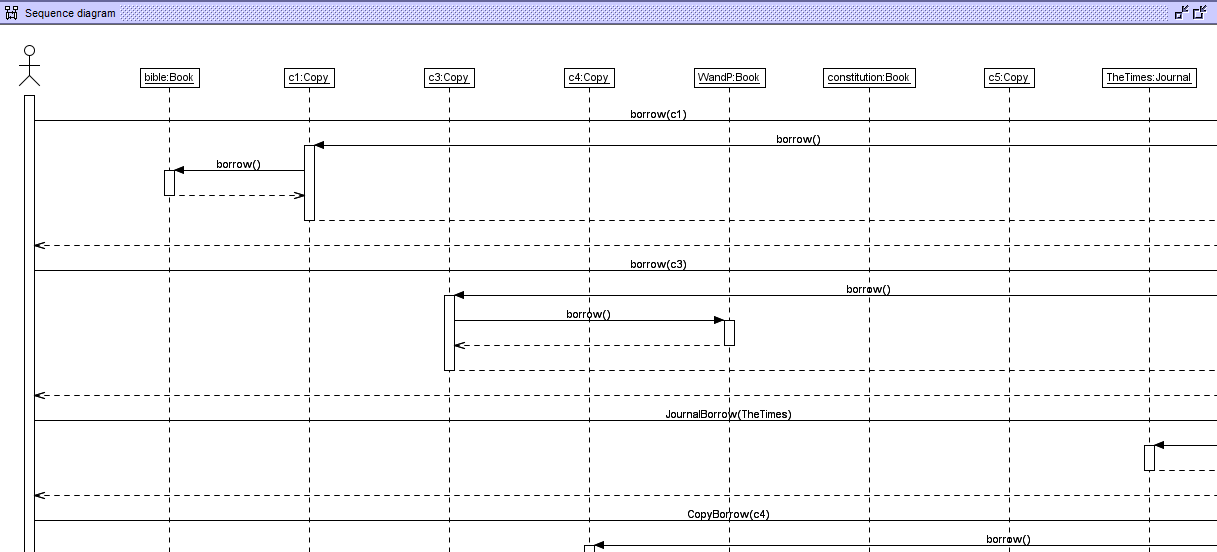
inv exist: book->size() > 0

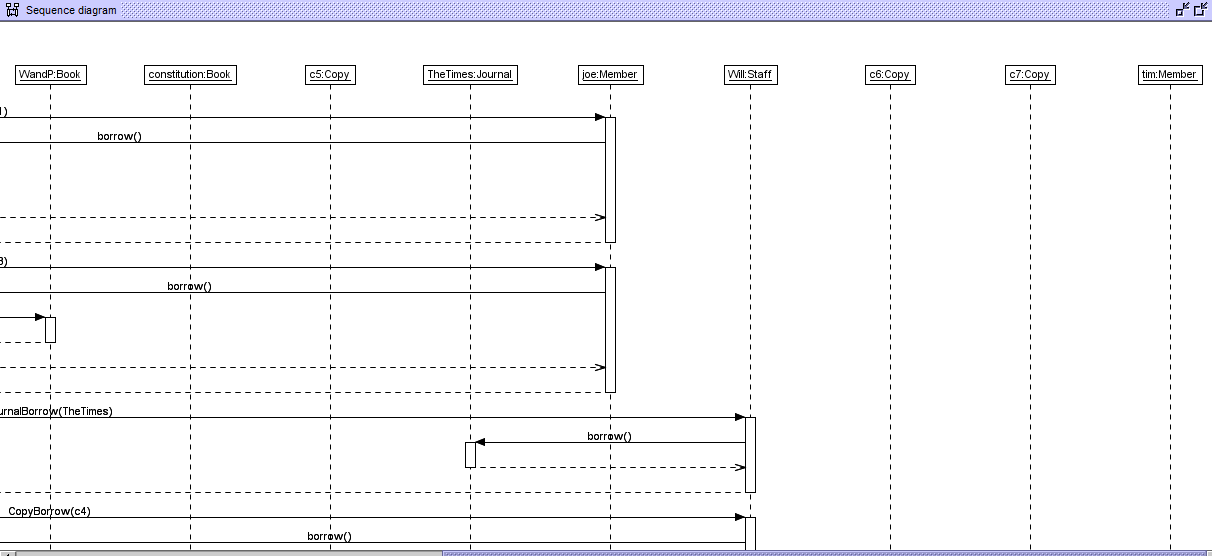
# Class Diagram

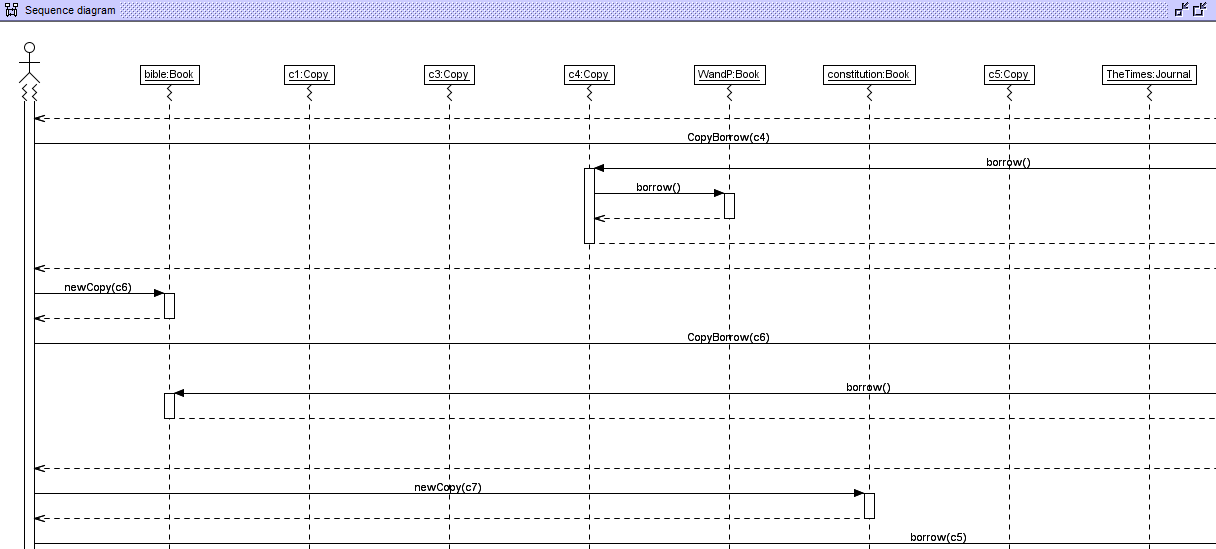


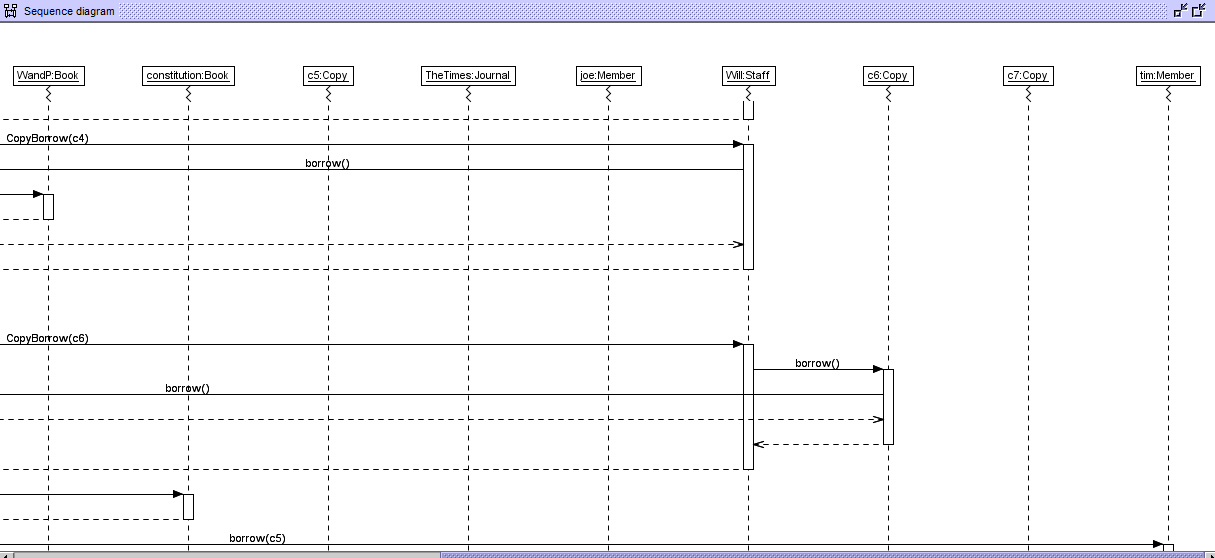
# Sequence Diagrams

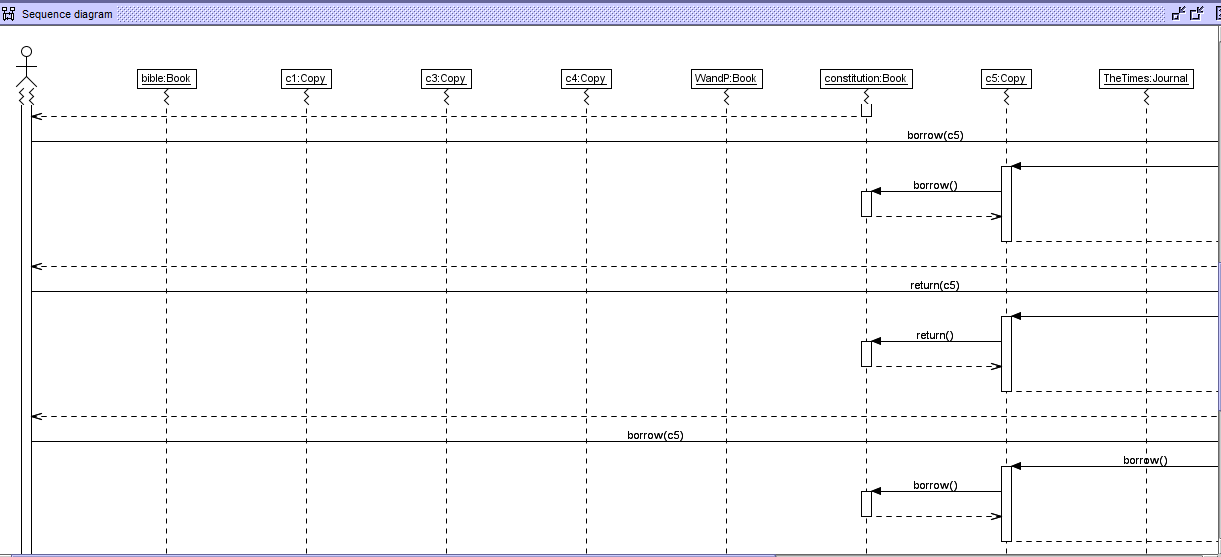
1)

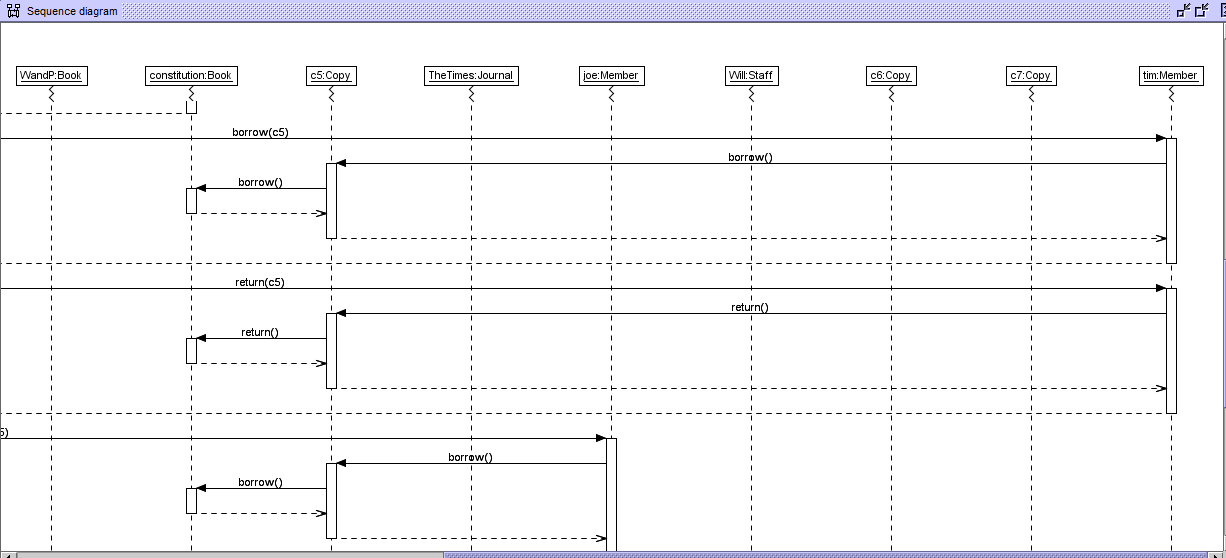


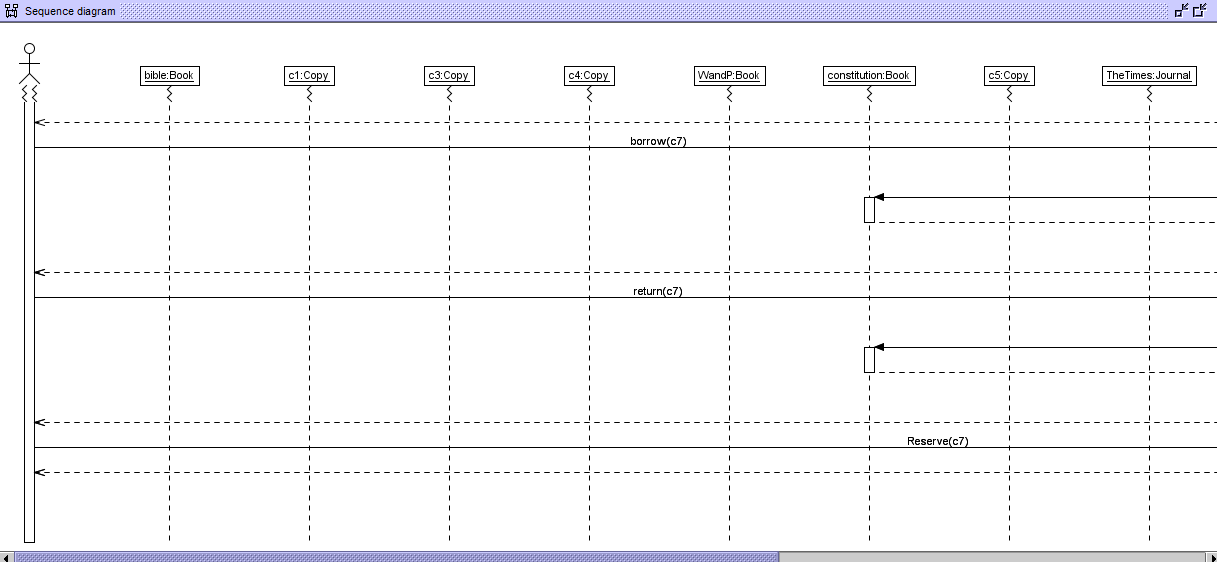


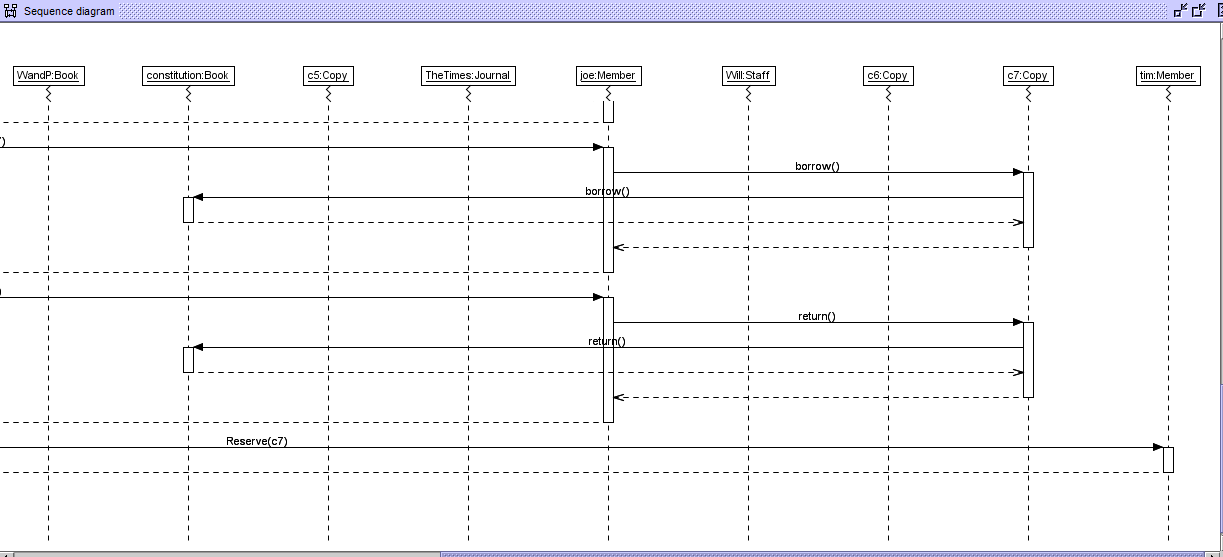




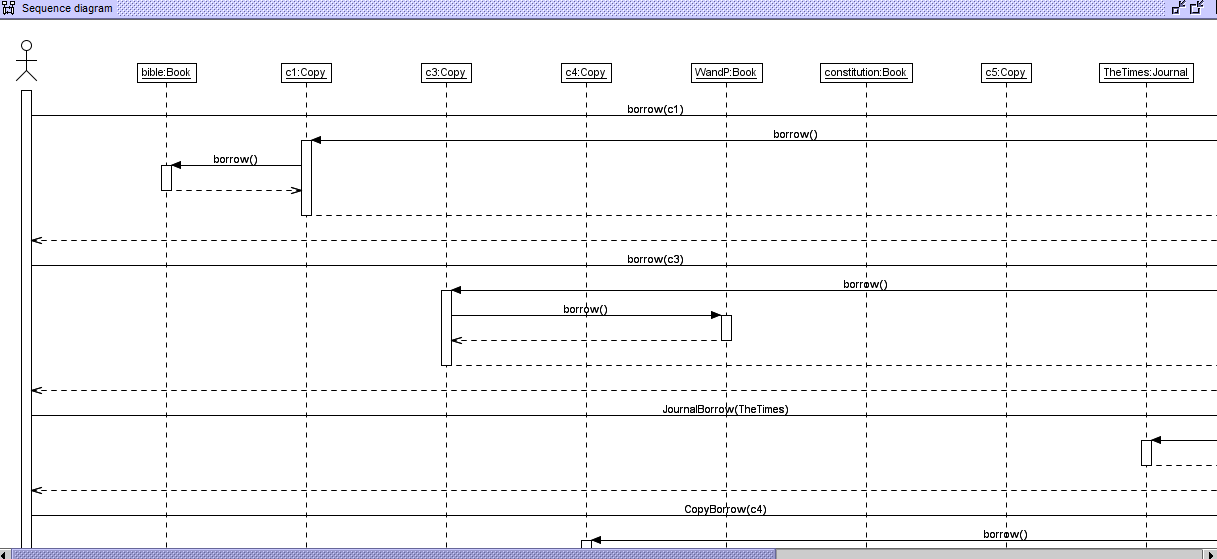


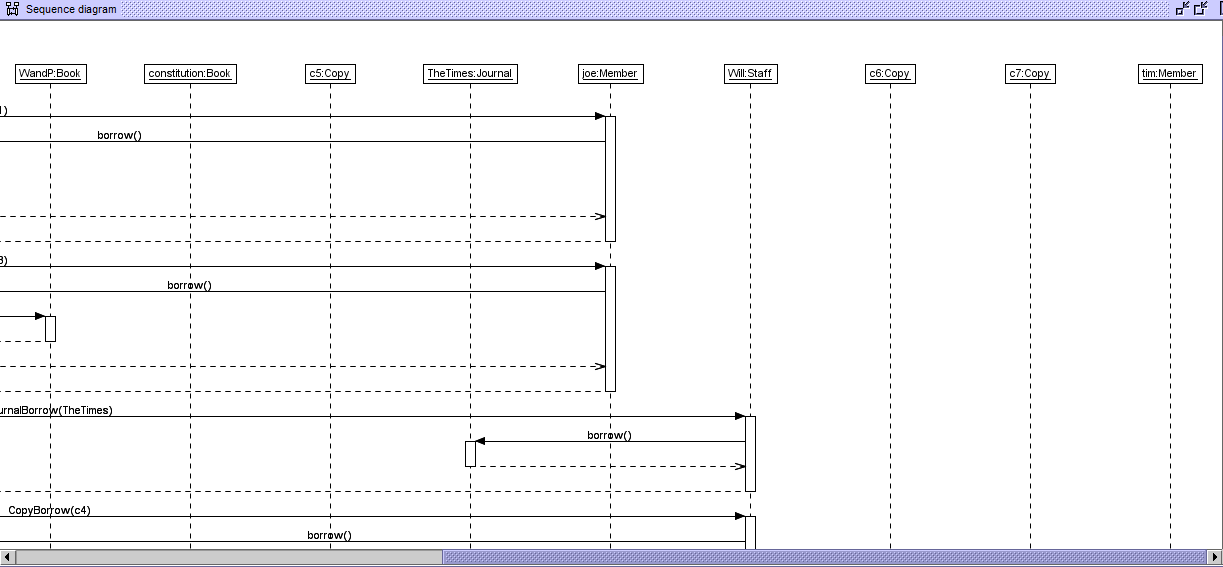


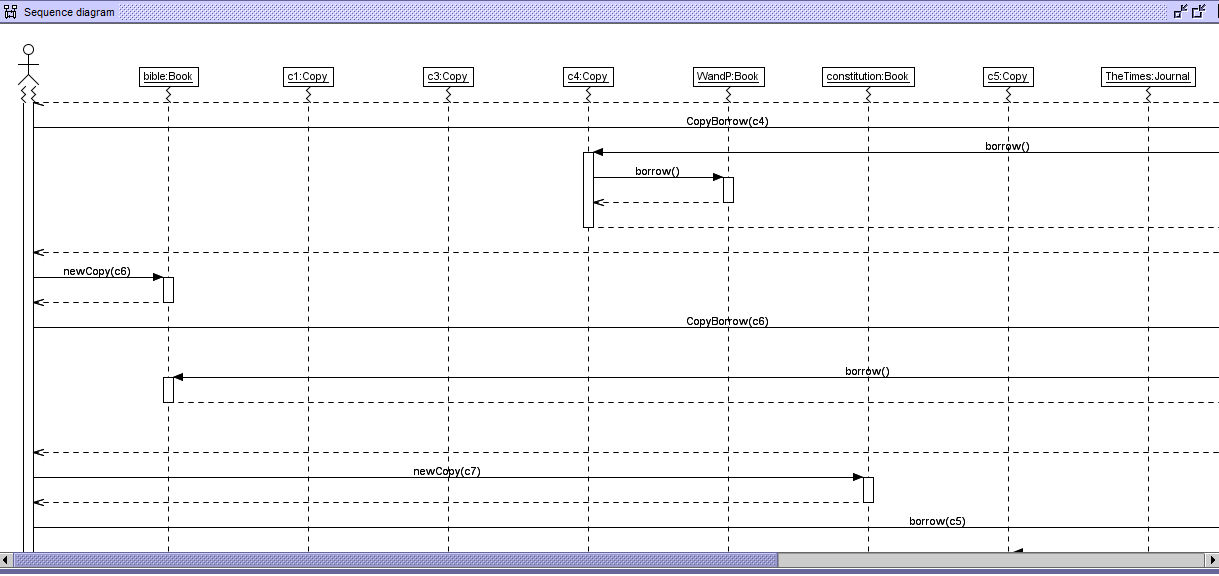


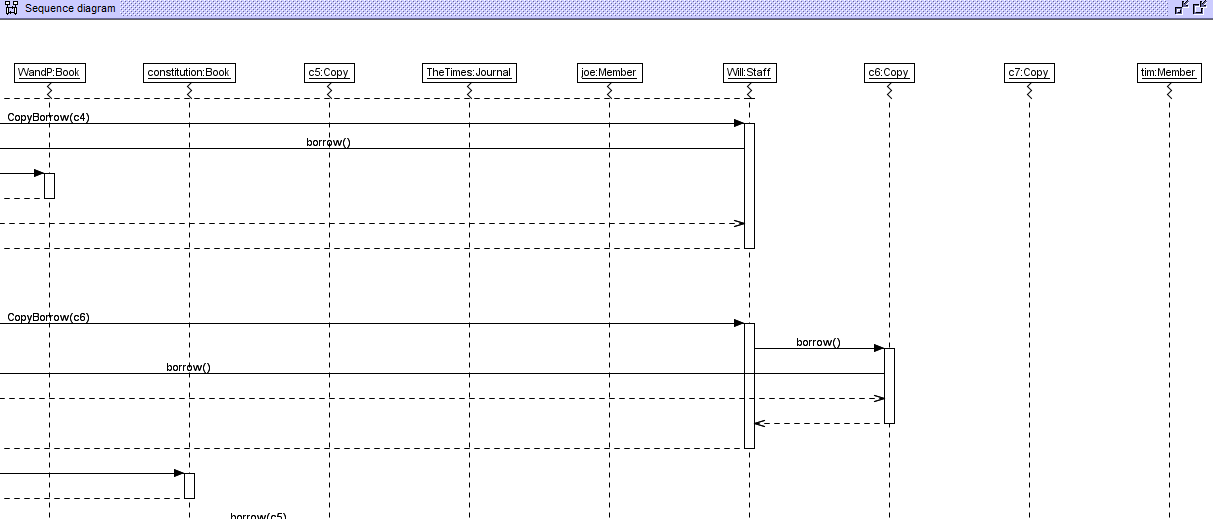


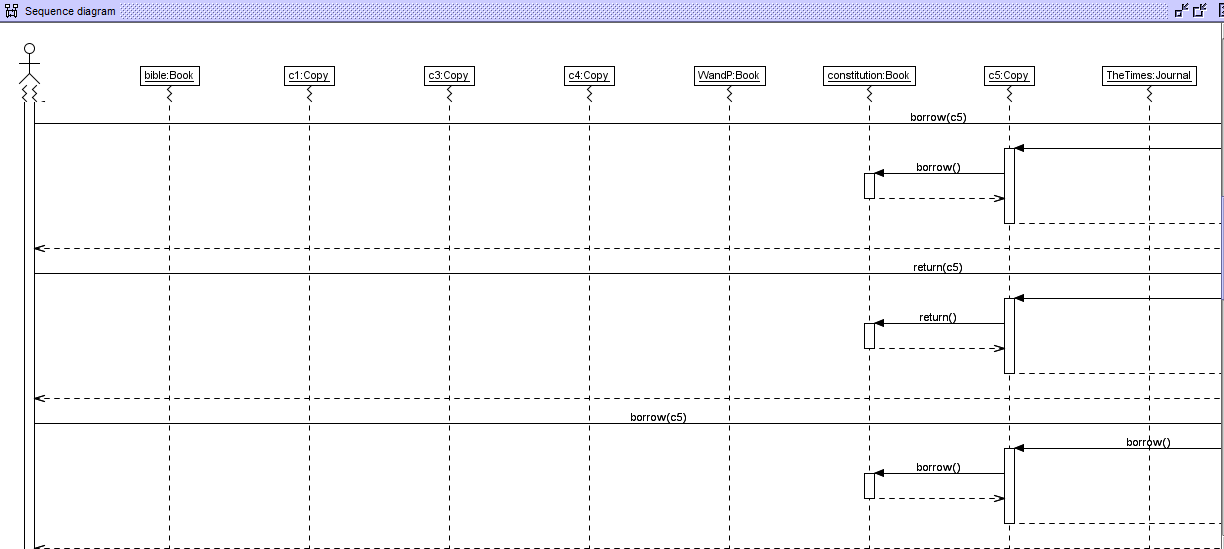
# 2)

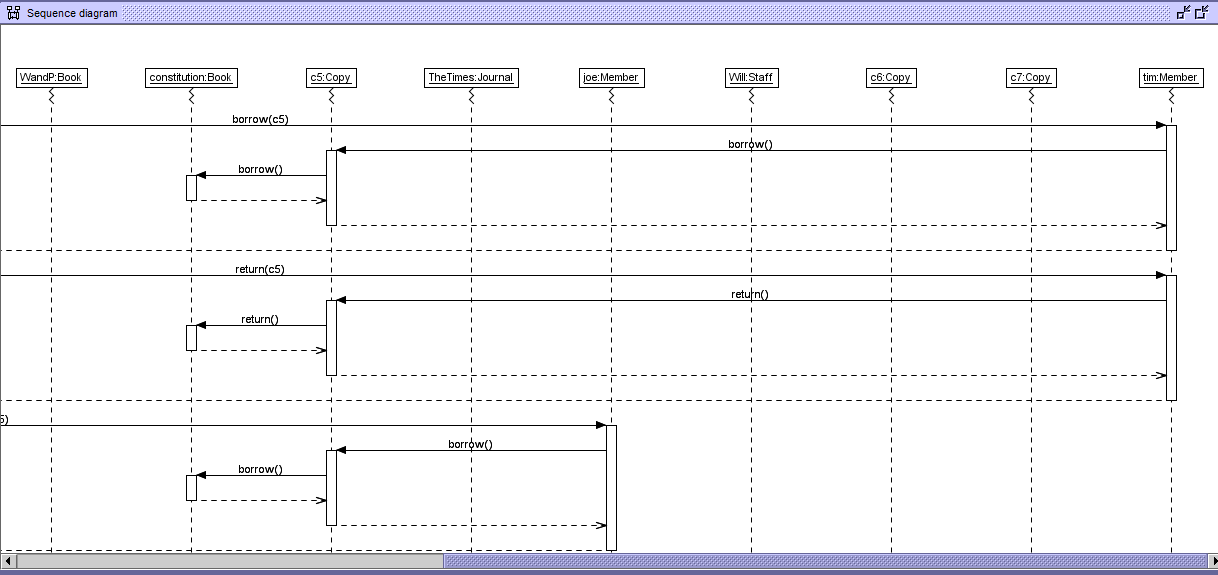


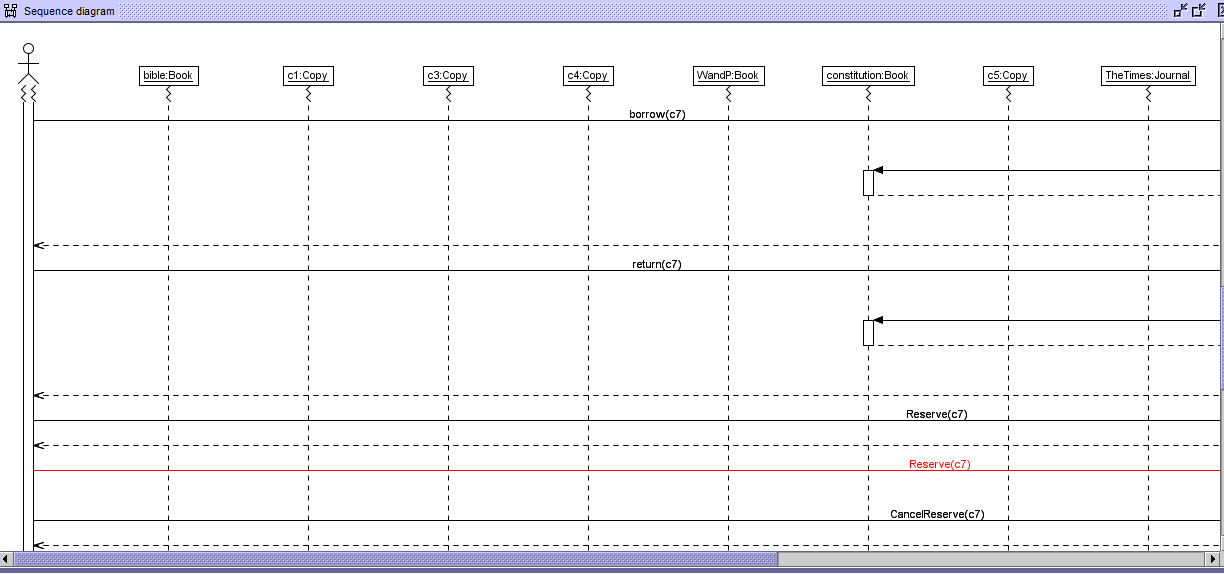


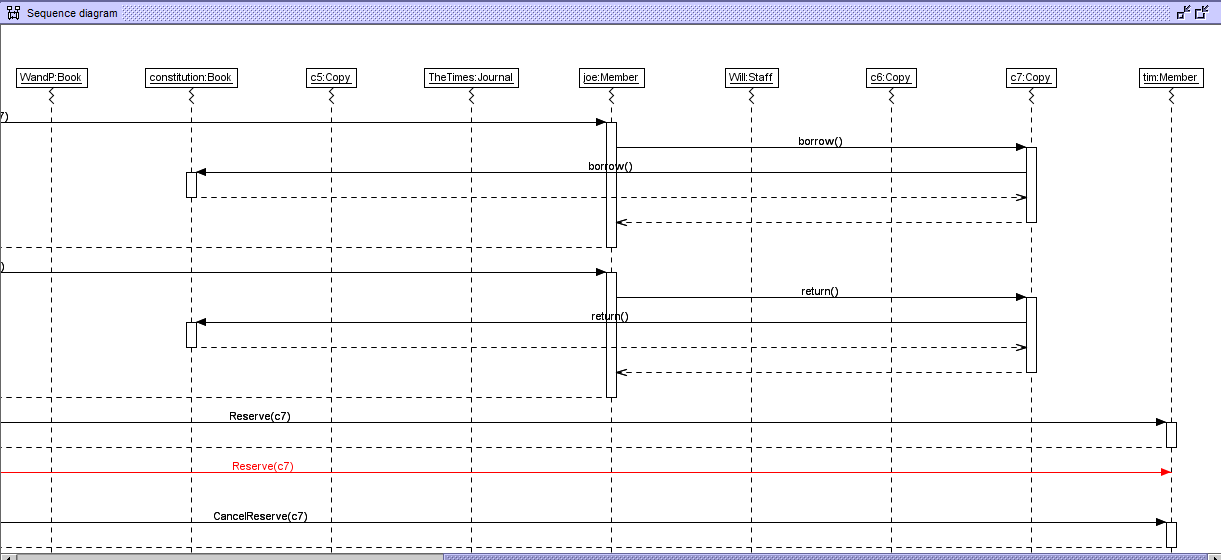


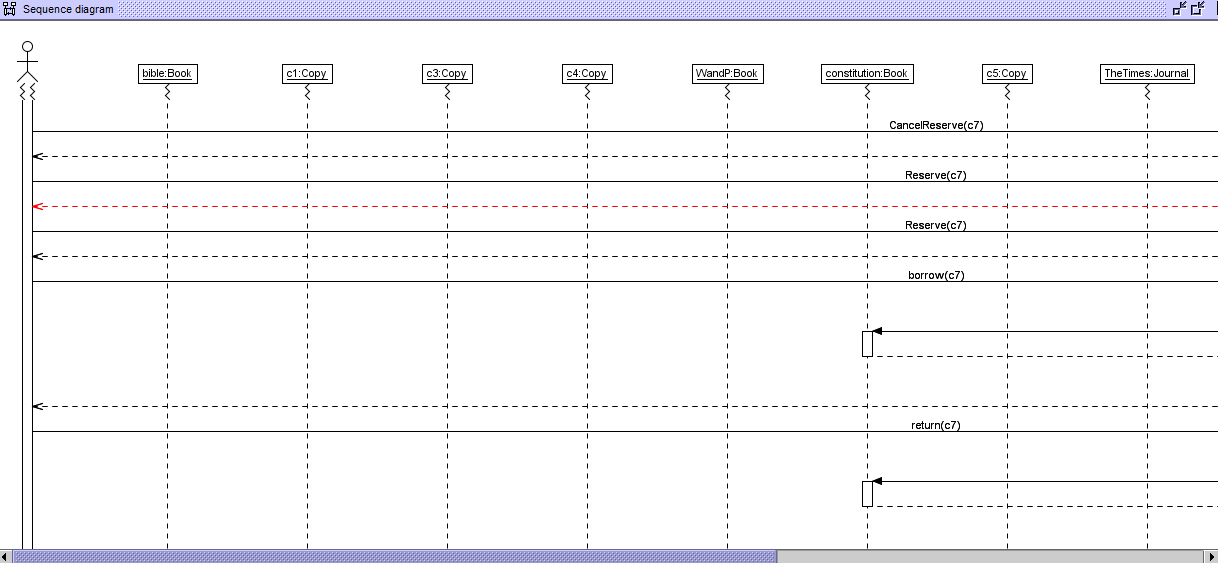


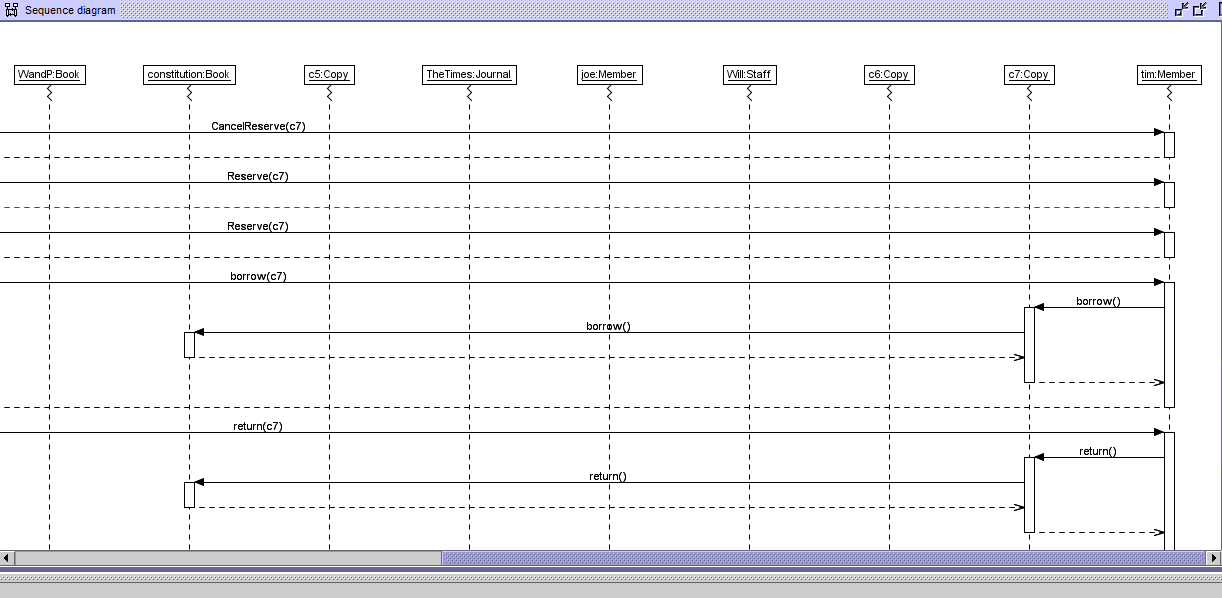


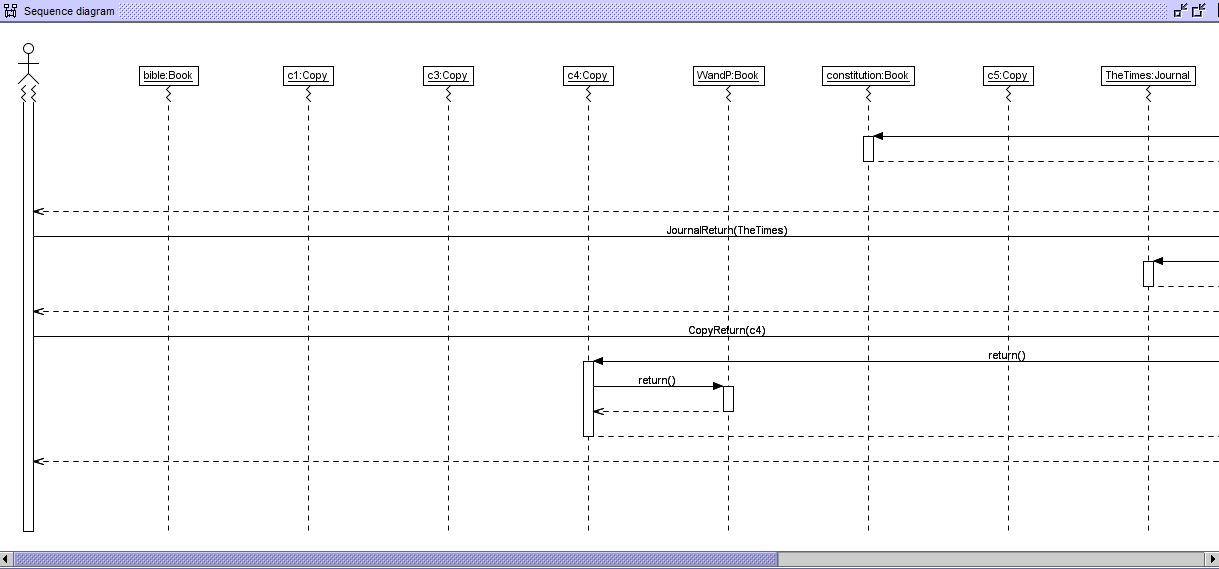


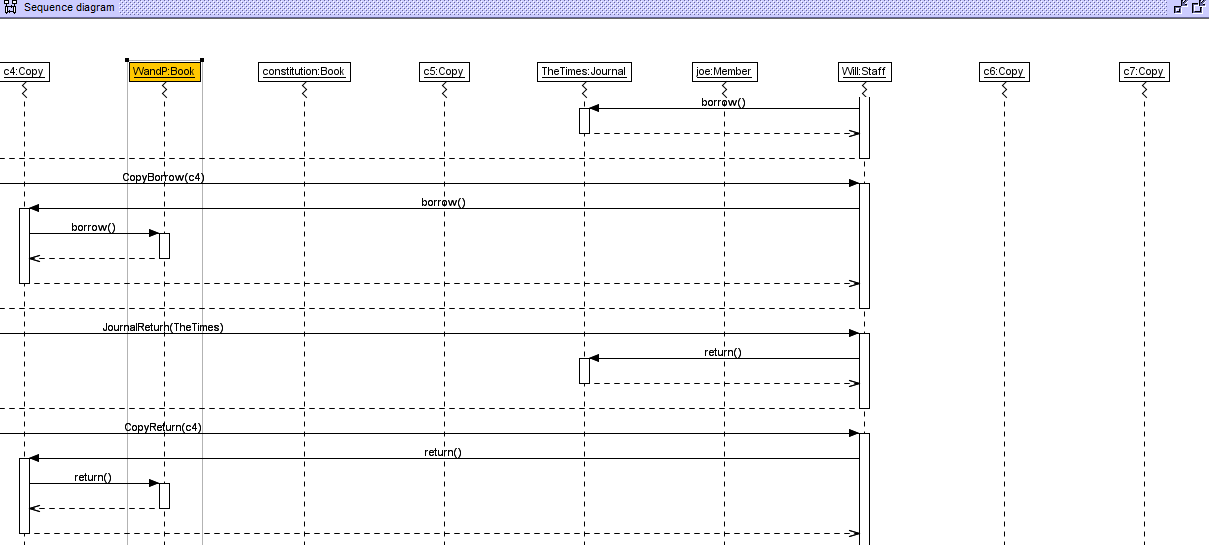




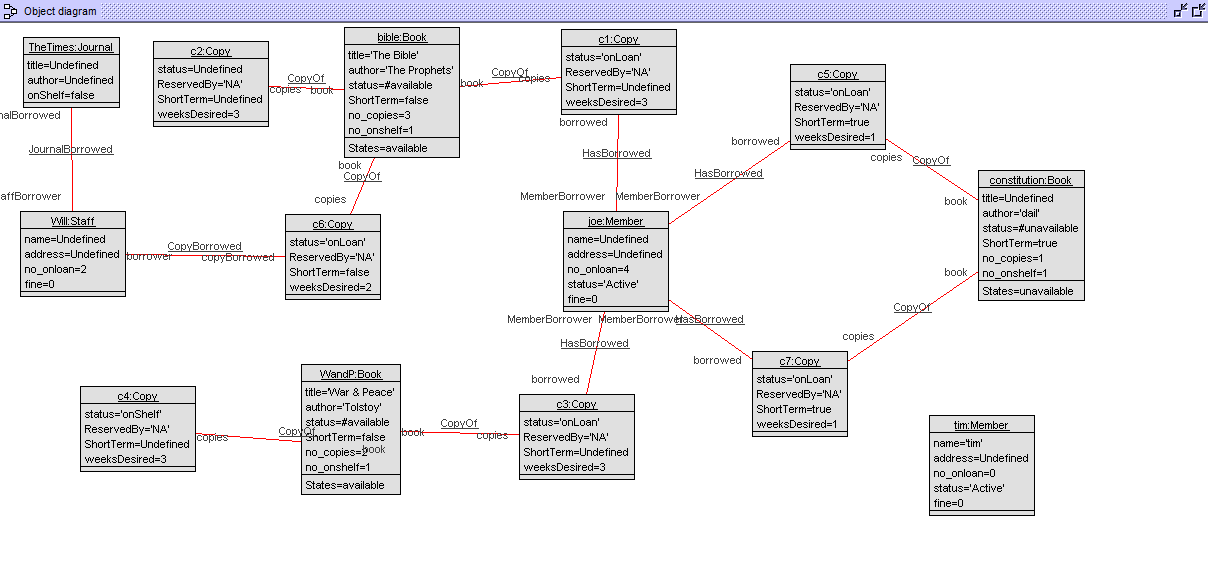
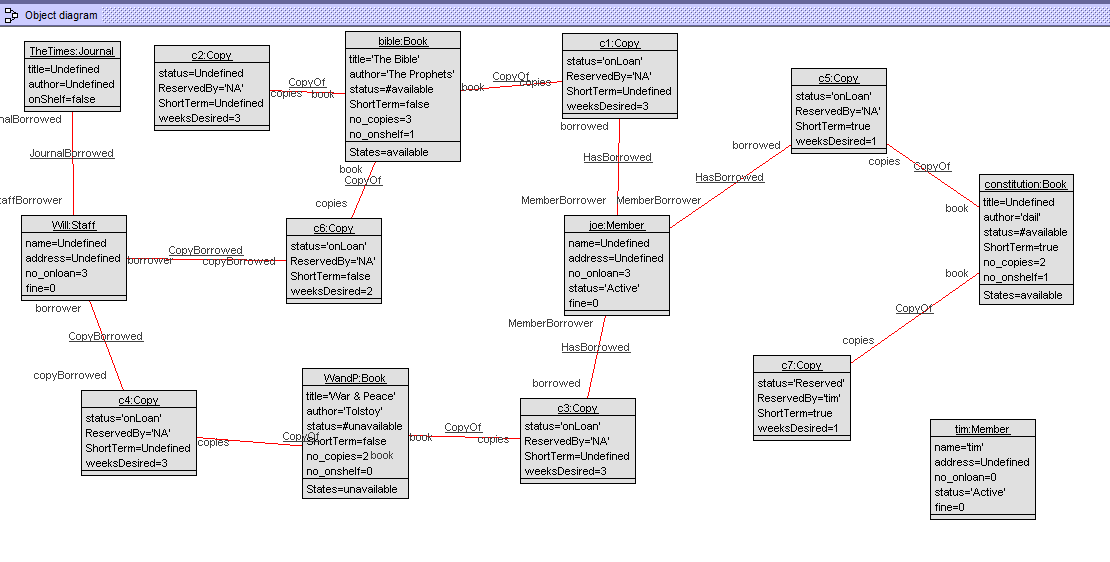




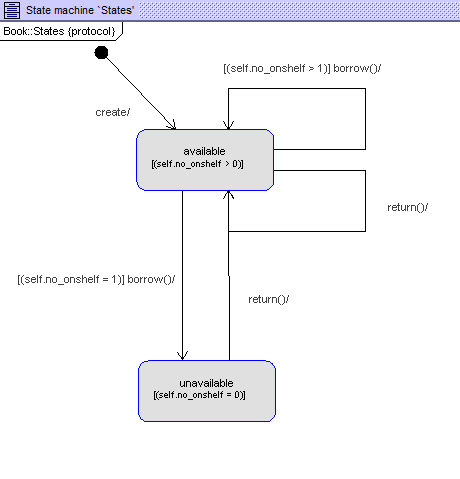




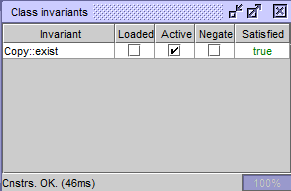
# Object Diagrams



# State Machine for Class Book

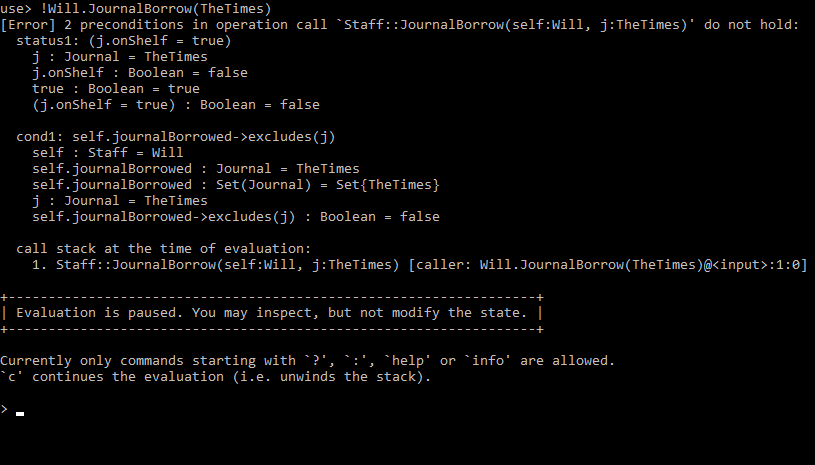


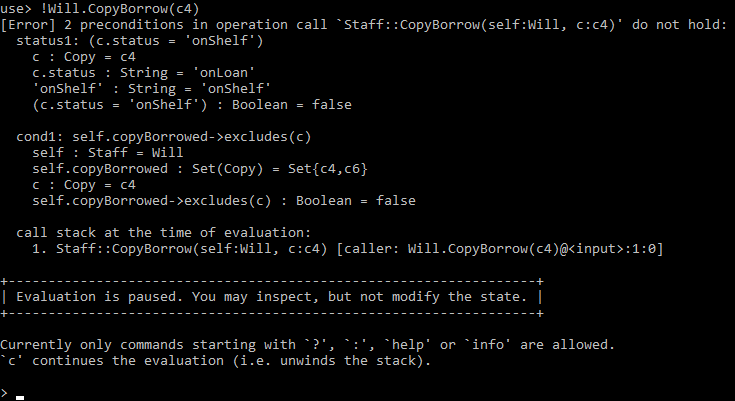
# Class Invariant:

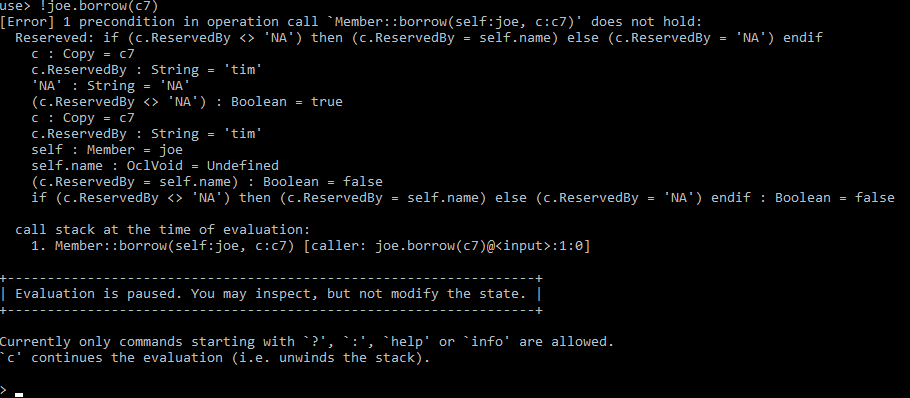


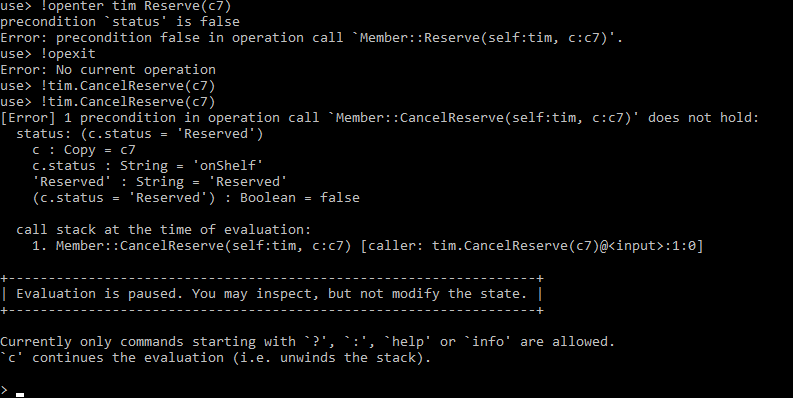
# Testing Constraints on the Command Prompt

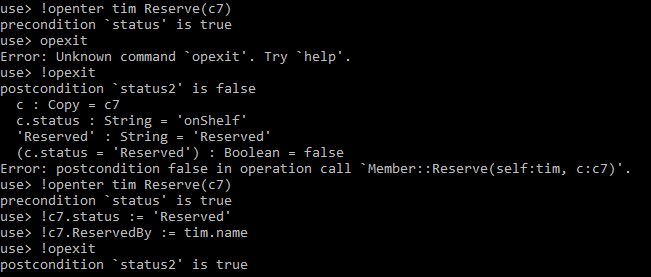
Screen Clipping

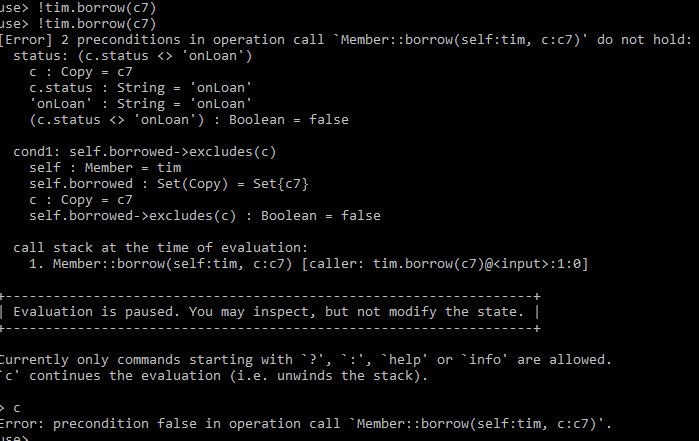


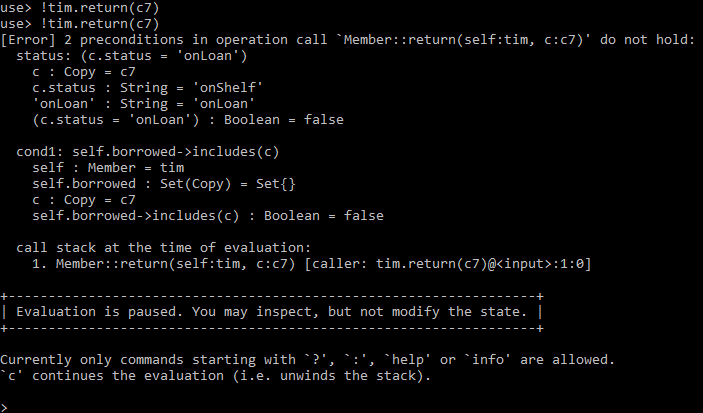


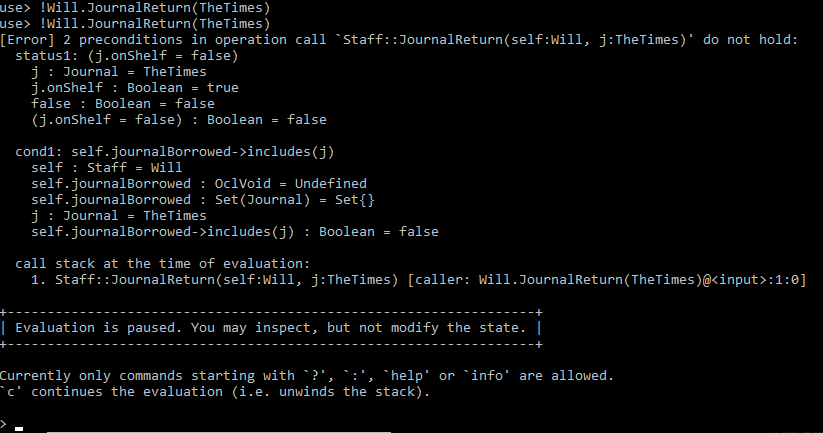


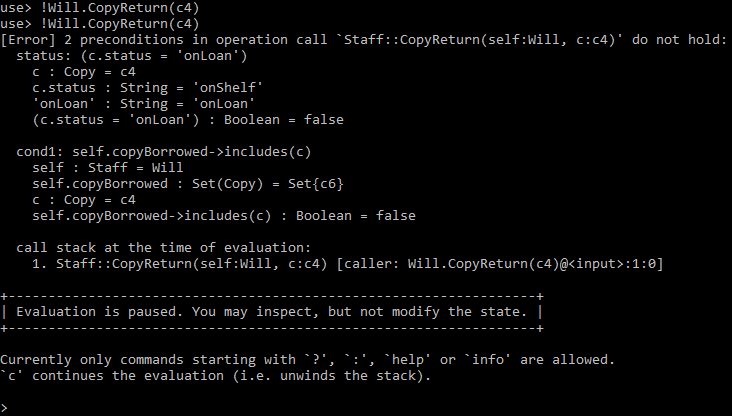












# Report on the Library Specification

## Introduction:

The report on an extension of the Library Specification that was discussed within class time. I will look at how the library model functions in accordance with how people would interact with the staff of the library and how the staff would interact with the system internally. I would also look at the limitations of the library specification and see if I get the expect results from building and testing the model.

We look at the different part. There is Journals and books in the library system. Each book has several copies associated with the book. Each journal does not have multiply copies themselves. The type of people who are interacting with the system are Members of the Library, who are ordinary people who have successfully applied for membership of the Library, and the staff of the library, the workers of the library.

The library system has its own way of loaning out the books and journals. Members are only able to borrow six books at any time, and only books. Members are not allowed to borrow journals. The staff can borrow the books and journals if they wish. Within a combination of both, Staff can borrow up to 12 at any given time. After both members and staff have passed their limitations, they cannot borrow any more copies.

We will build a system to ensure that the above requirements are met, and test them in accordance with possible questions such as

“Can Members of the Library and Staff members borrow Journals in this sytem?”

We wish to not allow such events to happen within the system as outlined in the specification given.

## Method:

We built the model in the language use and soil in uml. By building the objects, I would have a way of testing the capabilities of the system. To ensure the system followed the specifications, I built the appropriate constraints to ensure the system flowed in the expect way. After building all the classes and constraints, I built the associations to ensure there is a connection between each object that interacted with each other.

Once the building of the framework is completed, I build object using a soil file. This was the quickest way to test each object simultaneously. I would want to make sure that every connection, function and behaviours are appropriate before I try to test the faults of the system (if there was any). If I found a huge fault, I would immediately implement a constraint to prevent it from happening further down the line. I then check if this impacts any other part of the system that I wish it would not.

After each testing, I document the results so I knew what the behaviours are acting out as. If there was a flawed pattern, I would find it using the method of building and testing. The objects which I were testing would dynamically change depend on what I interact with. I would also look at the sequence diagram to see what exactly was happening within the programme. Some of the findings was screenshotted in this report above. Using the uml command line, I tested multiple objects and took screenshots of our findings and pasted them in this document. I also saved the testing as a text file to look back upon.

The class file associated with this work as well as the object file was laid out as visually pleasing for other users. The object file would be the file that dynamically change based upon our input. Once one object satisfied our requirements, I move onto the next object to test.

## Evaluation of Findings:

Summary of the findings example:

We will take the bible as an example. There are two copies available in this library. I will call them c1 and c2. There are two members, tim and joe. There is one staff, who is called Will. There is one journal which is known as the times. Will can borrow both both the bible (c1) and the times. So he does. tim comes along and requests to borrow a copy of the bible. He cannot borrow c1 since Will has already borrowed it. However there is c2, which he can borrow. The bible status remains to be available until c2 is taken from the shelf. After that, the status becomes unavailable.

Joe comes along and request a copy of the bible. However, He cannot take c2 nor reserve it at the moment since it is unavailable. However, when Will returns the bible, Joe can reserve it for himself to ensure no one else can take it. If he wants to release it back to open use, he can cancel the reservation so someone else can have the copy for themselves. These are the limitations of the system and the summary of the results.

Initially, the objects would be all over the place with each person be able to borrow an book that is already loaned, as well as reborrow the book that they already borrowed. To prevent this, the constraints implemented ensured that each journal and book can be only borrowed by one person at any time. Books available but only if they have a copy on the shelf ready to be borrowed.

The same rules applied to both staff and members. If a copy is already out, it cannot be borrowed. This is done by checking each copy availability. For example, the system has three words that it is switched from: “onShelf”, “Reserved” and “Loaned”. By comparing to these values, I can compare on whether each person can borrow the book. Once it is fully completed and fully tested, I continue to test to get to the result as the example explained briefly above. I went to ensure the system behaved as the specification requested.

## Conclusion:

The system was finally tested and deemed completed after all the implementations of both the use, soil and the diagrams documentation. After the testing has been completed, I went back to the panel which had all the ideas and made sure all the instructions have been followed. After that has been completed, I went back over the data to ensure the results are what I had expected. The first couple were not up to standard, but after fixing them, the results eventually came back to what was expected. The system could go on to be designed for more book genres and people specification ( such as newspapers and Managers) but the system designed was enough to do the satisfied work.