**Software Engineering 2 Continuous Assessment 2**

**Library System Report**

**Student Name:** Harry Delaney  
 **Student ID:** C23361046  
 **Developed with:** Cheong Zhi Yuan (D23124603) & Bohdan Yeriemienko (C23409402)

**Lecturer:** Mr. Richard Lawlor

**Module:** Software Engineering 2 CMPU 2020

**Project:** Library Management System in USE

**1. Introduction**

In this report, we explore the implementation and development of a Library Management system in USE. The system involves the management of books, copies and members, and the operations related to these objects within our system, such as borrowing, returning, reserving, paying and managing fines, and reporting books/copies damaged.

The report covers the following:

* System Design: An overview of the systems design and functionality
* Operations: Explaining the key operations within the system
* State Diagrams: Represents the states of books/copies after different operations
* Analysis & Discussion: Analyzing the design choices and discussion of the system and its implementation.

**2. System Overview**

The Library System consists of the following 3 classes: Book, Copy and Member.

* Book: Represents a book in a library e.g. Harry Potter, with the attributes (title, author, status, no\_copies & no\_onshelf). The statuses include (available, unavailable and onReserve)
* Copy: Represents individual copies of books from our library with the attribute (status). These statuses include (onShelf, onLoan, onReserve and damaged).
* Member: Represents a member of the library with the attributes (name, address, no\_onloan, status and fine).

Member Functionality:

* Borrow: A member can borrow a book if it is available
* Return: A member can return a book they borrowed
* Reserve: A member can reserve a book
* Unreserve: The member can unreserve the book which they reserved
* Pay a Fine: A member can pay an outstanding fine, so that they can borrow or reserve books again
* Report Damaged: A member can report a book damaged, making that copy unavailable.

**3. Use Cases:**

For the Library USE model, the provided original model has borrow() and return() as use cases. In our improved model, we added reserve(), unreserve(), payFine(), and reportDamaged() as new use cases to the model. Below outline the logic of the use cases and how they work.

1. **Reserving a book:**

* USE case: Reserve()
* Actors: Member
* Description: The use case allows a member to reserve a copy of a book, making it so no other member can reserve/borrow that copy
* Pre Conditions: The book must be eligible for reservation i.e. status = #onshelf.

The member must have no outstanding fines, and fewer than 2 books borrowed.

* Events:

1. Member selects the copy of the book they want to reserve
2. If eligible, the member reserves the book.
3. Number of available copies (no\_onshelf) decreases by 1
4. Status of copy is updated to onLoan
5. If no more copies are available, the books status is updated to unavailable

**2.** **Unreserving a book:**

* USE case: Unreserve()
* Actors: Member
* Description: Allows the user to cancel a reservation of a copy of a book they had, making it available again to library members.
* Pre Conditions: The member must have a currently reserved copy of a book
* Events:

1. Member Unreserves their currently reserved copy
2. The system updates the copies status to onShelf
3. The number of available copies of the book is increased by 1
4. If the book was unavailable before unreserving, its status is updated to available.

**3.**  **Paying a fine:**

* USE case: payFine()
* Actors: Member
* Description: Allows a Library member to clear their profile of a fine, which was preventing them from borrowing or reserving books.
* Preconditions: The Member must have an outstanding fine i.e. (fine > 0)
* Events:

1. The member pays their fine
2. The system subtracts the amount paid by the user from their fine total.
3. If the amount paid brings their fine total to 0, they are eligible to once again borrow and reserve books.

**4. Reporting a damaged book:**

* USE case: reportDamage()
* Actors: Member
* Description: Allows a user to report a copy of a book damaged, making it unavailable.
* Preconditions: The copies status must be onShelf.
* Events:

1. The member identifies a damaged copy.
2. The member reports it as damaged
3. The status of the copy is updated to damaged.
4. The copy is now no longer available to be borrowed or reserved.
5. The number of available copies of the book decreased by 1.

**4. Class Design**  
Below will show an in-depth look at the design for each of our three classes, with diagrams to aid the explanation.

Class diagram:

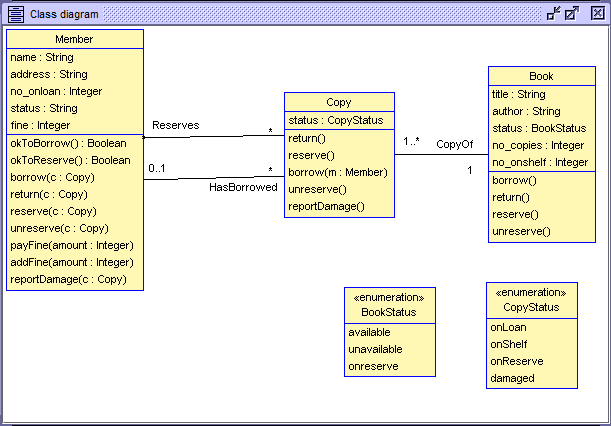
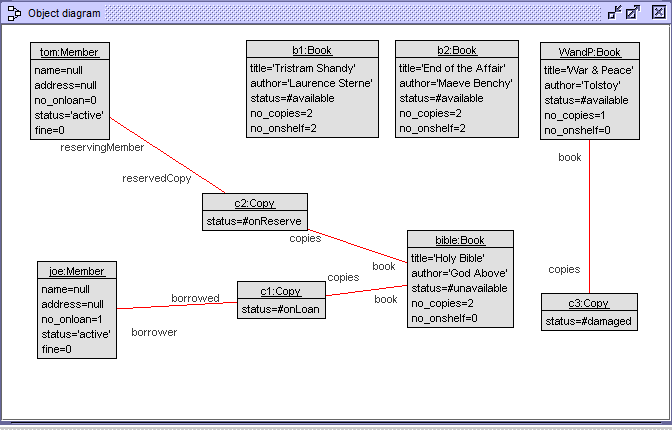


Figure 4.0.1: Class Diagram

Object Diagram (with various copy statuses):



4.0.2: Object Diagram

**4.1 Book Class**

The book class has the book's title, status, total number of copies of the book, and the number of those copies available. It includes the following operations

Operations:

* Borrow(): Reduces the number of available copies by 1 and updates the status
* Return(): Increases the number of available copies by 1 and updates the status.
* Reserve(): Marks the copy reserved, meaning it cannot be borrowed by another member. Also reduces the number of available copies by 1.
* Unreserve(): Marks the reserved copy as now available and increases the number of available copies by 1.

Book Class USE code:

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;  
 if (self.no\_onshelf > 0 and self.status = #unavailable) then  
 self.status := #available  
 end  
 end  
 post: no\_onshelf = no\_onshelf@pre + 1  
  
 reserve()   
begin  
 self.no\_onshelf := self.no\_onshelf - 1;  
 if (self.status = #onShelf) then  
 self.status := #onreserve  
 end  
 end  
  
 unreserve() begin  
 -- Logic to change status back from onreserve.  
 -- Should consider if any copies are now on shelf.  
 self.no\_onshelf := self.no\_onshelf + 1;  
 if (self.no\_onshelf > 0) then  
 self.status := #available  
 else  
 self.status := #unavailable  
 end  
 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

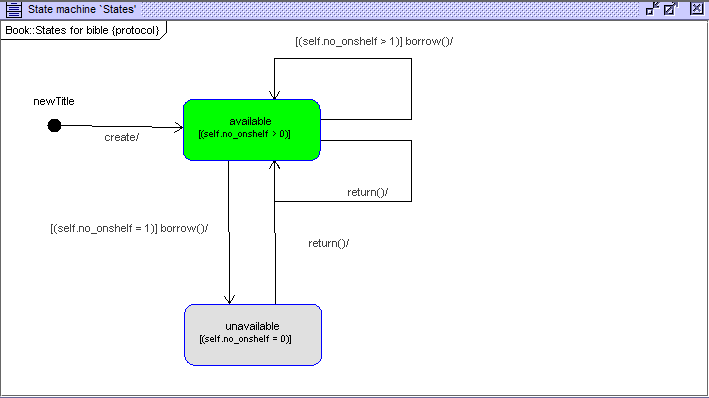
State Machine Diagram  


Figure 4.1.1 Book State is available

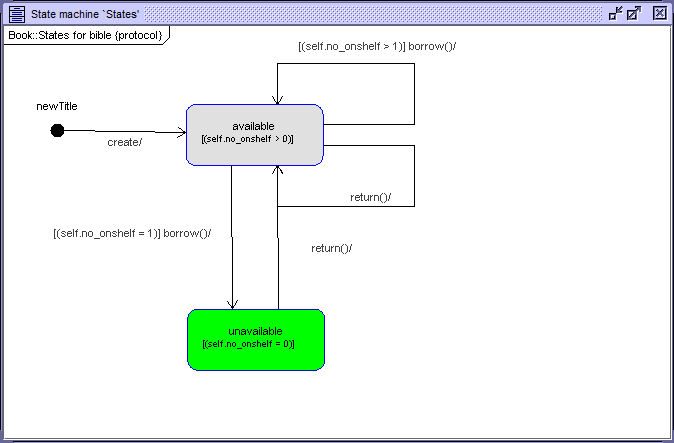


Figure 4.1.2 Book state is unavailable

**4.2 Copy Class**

The copy class manages individual copies of a book, with the following operations:

Operations:

Borrow(): Borrows the copy, changes the status of the copy to unavailable, and reduces the number of copies onshelf by 1

Return(): Returns the Copy, changing its status to available, and increases the number of copies onshelf by 1.

Reserve(): Reserves the copy, the copies status to onLoan, and reduces the number of copies on shelf by 1.

Unreserve() Unreserves the copy, changing its status to available, and increases the number of copies onshelf by 1.

Copy Class USE code:  
class Copy

attributes

status : CopyStatus init = #onShelf

operations

return()

begin

self.status := #onShelf;

self.book.return() end

reserve()  
 begin  
 if (self.status = #onShelf) then  
 self.status := #onReserve;  
 self.book.no\_onshelf := self.book.no\_onshelf - 1;  
  
 if (self.book.no\_onshelf = 0 and self.book.status = #available) then  
 self.book.status := #unavailable;  
 end  
 end  
 end  
  
 borrow( m : Member)  
 begin  
 self.status := #onLoan;  
 self.book.borrow()  
 end  
  
 unreserve()  
 begin  
 if (self.status = #onReserve) then  
 self.status := #onShelf;  
 self.book.no\_onshelf := self.book.no\_onshelf + 1;  
  
 if (self.book.status = #unavailable and self.book.no\_onshelf > 0) then  
 self.book.status := #available;  
 end  
 end  
 end  
  
reportDamage()  
begin  
 if (self.status = #onShelf or self.status = #onReserve) then  
 self.status := #damaged;  
 self.book.no\_onshelf := self.book.no\_onshelf - 1;  
 end  
end  
  
statemachines  
 psm States  
 states  
 newCopy : initial  
 onLoan  
 onShelf  
 onReserve  
 --damaged  
 transitions  
 newCopy -> onShelf { create }  
 onShelf -> onLoan { borrow() }  
 onLoan -> onShelf { return() }  
 onShelf -> onReserve { reserve()}  
 onReserve -> onLoan { borrow() }  
 onReserve -> onShelf { unreserve() }  
 --onShelf -> damaged { reportDamage() }  
 --onLoan -> damaged { reportDamage() }  
 --onReserve -> damaged { reportDamage() }  
 end

End

State machine diagrams:

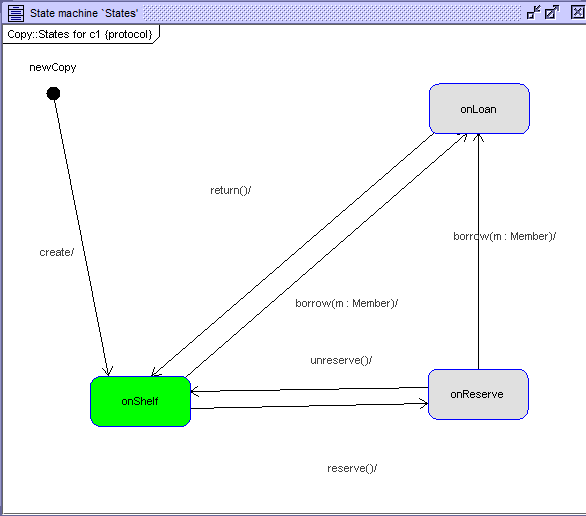


Figure 4.2.1: Copy state diagram in its ‘initial’ state, being onShelf

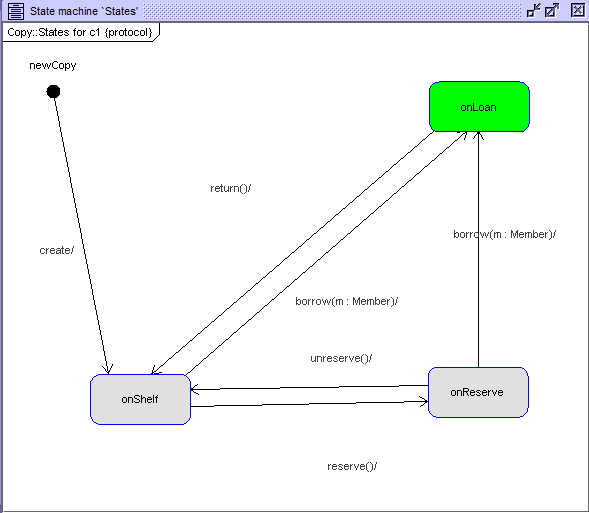


Figure 4.2.2: Copy status is now onLoan

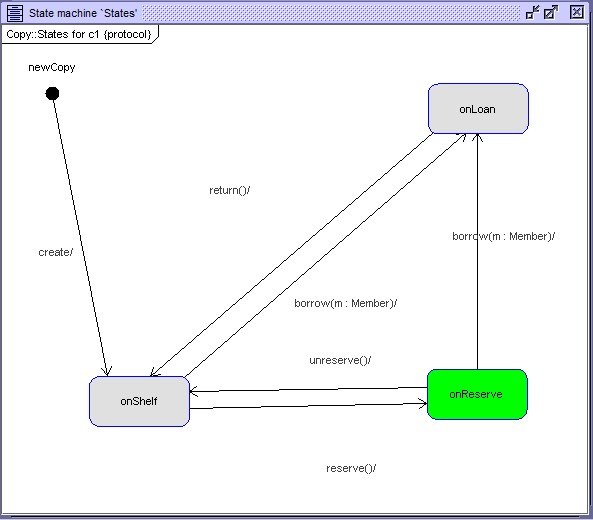


Figure 4.2.3: Copy status is now onReserve

**4.3 Member Class**

This class represents library members, allowing them to do all the actions discussed in the previous classes, and adds additional functionality.

Operations:

* OkToBorrow(): Checks if the member has less than 2 books currently onLoan, and that they have no outstanding fines.
* Borrow(): A member borrows a book, it adds that book to their HasBorrowed, and their number of on loan books increases by 1
* Return(): Removes the copy of the book from their HasBorrowed, and decreases their number of books onloan by 1.
* Reserve(): Inserts the copy of the book into the members Reserves.
* Unreserve(): Removes the copy of the book from their Reserves.
* PayFine(): Removes the amount the member pays from their fine total
* Addfine(): Can add an amount to the fine total. \*This operation was for testing purposes\*.

Member Class USE code:

class Member

attributes

name : String

address : String

no\_onloan : Integer init = 0

status : String init = 'active'

fine : Integer init = 0

operations

okToBorrow() : Boolean

begin

if (self.no\_onloan < 2 and self.fine = 0) then

result := true

else result := false

end

end

okToReserve() : Boolean  
begin  
 if (self.no\_onloan < 2 and self.fine = 0) then  
 result := true  
 else  
 result := false  
 end  
 end  
  
 borrow(c : Copy)  
 begin  
 declare ok : Boolean;  
 ok := self.okToBorrow();  
 if( ok and c.status <> #damaged ) then -- updated line  
 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  
 if (c.status = #onShelf) then  
 insert (self, c) into Reserves;  
 c.reserve();  
 else  
  
 end  
end  
  
 unreserve(c: Copy)  
 begin  
 delete (self, c) from Reserves;  
 c.unreserve();  
 end  
  
payFine(amount : Integer)  
begin  
 if (amount >= self.fine) then  
 self.fine := 0;  
 -- Optionally, you can record the fine amount  
 else  
 self.fine := self.fine - amount;  
 end  
end  
  
addFine(amount : Integer)  
begin  
 self.fine := self.fine + amount;  
end  
  
reportDamage(c : Copy)  
begin  
if (c.status = #onShelf or c.status = #onReserve) then  
 if c.status = #onShelf then  
 c.book.no\_onshelf := c.book.no\_onshelf - 1;  
 end;  
 c.status := #damaged;  
end  
end

end

**4.4 Sequence Diagram of different system interactions**

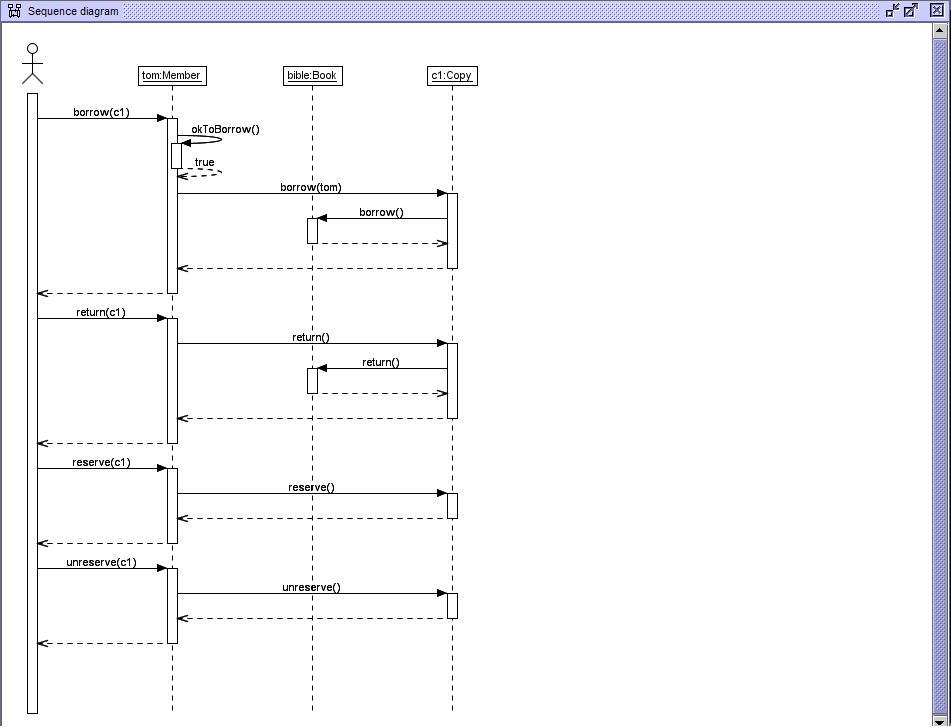


Figure 4.4.1: A sequence diagram of a member borrowing and then returning a copy, then reserving and unreserving another copy.

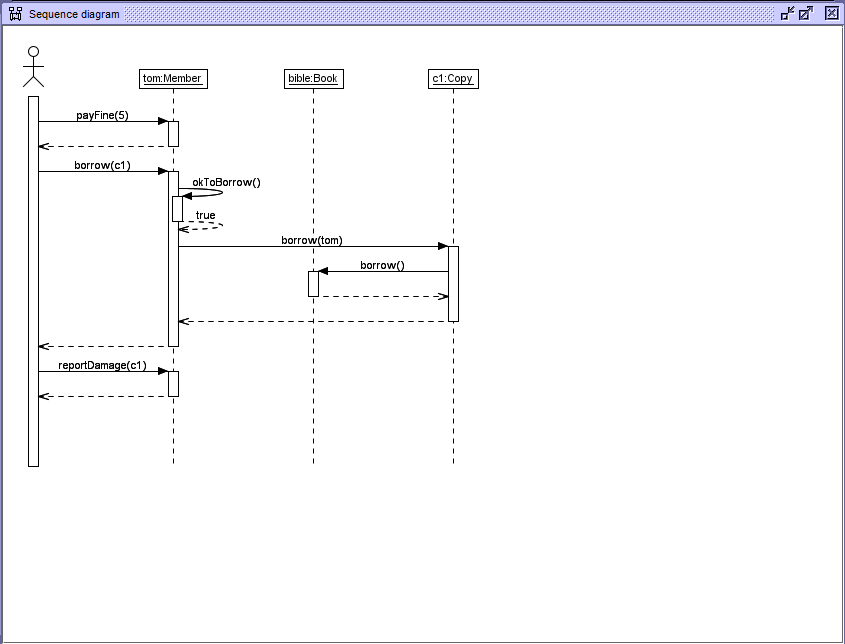


Figure 4.4.2: A library member pays their outstanding fine, then borrows a copy of a book, then reports that copy damaged.

**5. Constraint Testing:**

For the following testing of the different use cases, each time the following SOIL file was ran, to create the objects:

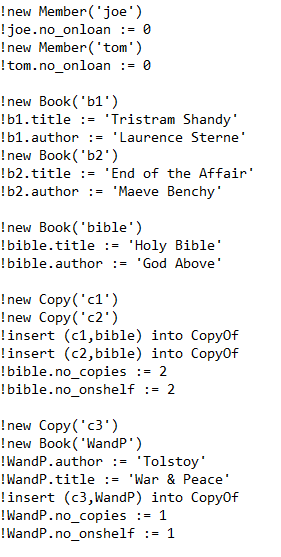


Figure 5.0.1: SOIL testing file contents.

**5.1 Borrow Constraints:**  
Trying to borrow the same copy of a book again:

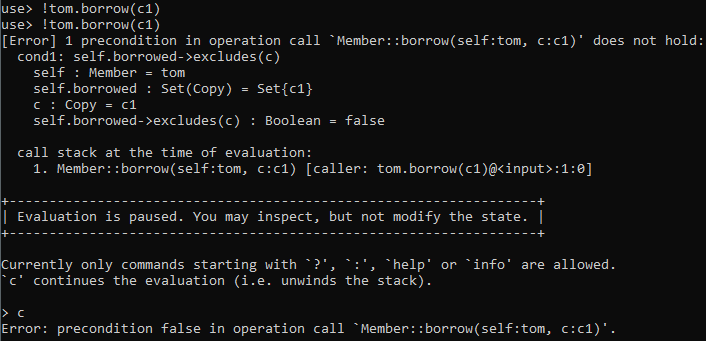


Figure 5.1.1: Terminal output of the above test

Trying to borrow a book when the loan limit has been reached:

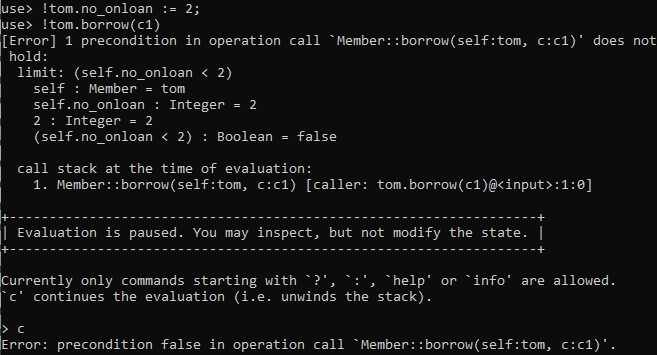


Figure 5.1.2: Terminal output of the above test

Trying to borrow a book with a fine:

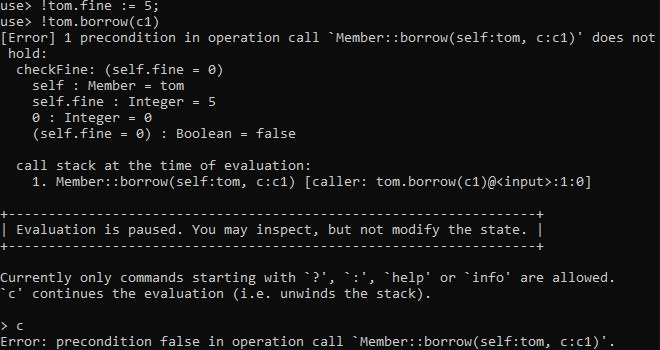


Figure 5.1.3: Terminal output of the above test

**5.2 Reserve Constraint testing**

!Openter and !Opexit for Reserving:

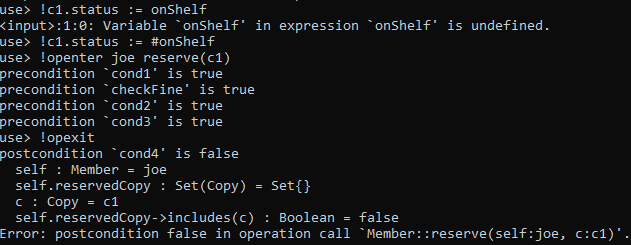


Figure 5.2.1: !Openter and !Opexit testing of the reserve class

**6. Discussion and Analysis**

**6.1 USE case choices:**

* The decision to implement the new use cases of reserving, unreserving, reporting damage and paying fines was to add a new level of detail to the system.
* The ability to reserve books allows library members to set aside a copy, without it being accessible to others.
* The fine system is integral to ensuring members return books and enforcing rules in the library
* Reporting books damaged allows members to help each other, by reporting a book damaged, it is no longer available to be borrowed or reserved, meaning another member won't mistakenly borrow or reserve it.

**6.2 Design Choices:**

* The decision behind the states, such as available and unavailable, was to allow for accurate tracking of book availability within the library.
* The state transitions for the classes were handled carefully to ensure a logical and efficient flow.
* The utilization of state diagrams aided greatly in visualizing state transitions and what actions were causing them.

**6.3 Constraints:**

The library system enforces preconditions and invariants to ensure the library operations follow consistent and logical rules. We saw these tests earlier in the report. Below is the list of constraints with a brief description

Borrowing Constraints:

* Borrow limit: No more than 2 books can be borrowed.
* Fine Check: A member cannot have an outstanding fine.
* Cannot reserve the same book again: Members cannot reserve books they already have borrowed.
* Post-Condition: The borrowed copy is added to the member’s borrowed list.

Borrow class Constraint USE code:

context Member::borrow(c:Copy)

pre limit: self.no\_onloan < 2

pre checkFine: self.fine = 0

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

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

Reservation Constraints:

* Reservation Eligibility: Members must have less than two books borrowed.
* Fine Check: Member cannot have an outstanding fine.
* Cannot reserve a borrowed book: A member cannot reserve a book they have borrowed.
* Duplicate Reservation check: Cannot reserve the book they already have reserved.
* Post-Condition: The reserved copy is added to the members reserved list.

Reserve Class Constraint USE code:

context Member::reserve(c:Copy)

pre cond1: self.no\_onloan < 2

pre checkFine: self.fine = 0

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

pre cond3: self.reservedCopy->excludes(c)

post cond4: self.reservedCopy->includes(c)

Additional Book Constraint:

* Number of copies: There must be more than 0 copies of the book .

These constraints collectively ensure that there is a logical and fair rule set to the library, and that the rules cannot be violated. They also allow for consistency by tracking books and copy statuses and numbers.

**6.4 Current Limitations and Future Improvements**

* Currently there is no set way of determining a fine for a member. It is just added to their fine total. Future prototypes could include fine calculations based off how overdue a book is.
* The system for reporting damaged books can be expanded, for example, some books may be beyond repair, whereas others may just have water marks. Being able to differentiate between them will help in the future.
* Reserving books can be implemented more in depth. For example, the ability to pre-reserve a book so that when it becomes available you get it first, and a queue, of course, would be added to this.

**7. Conclusion**

The library system described throughout this report provides essential key features a library system would be expected to have, for managing books, copies and members. Simple and logical borrowing, reserving, returning and unreserving etc., of books, along with the handling of fines offer a great system, with the ability to be improved on and expanded greatly in the future.

**8. References**

* What is a State Machine Diagram, Visual Paradigm (2019)
* GeeksForGeeks, Use Case Diagram for Library Management System,

Retrieved from https://www.geeksforgeeks.org/use-case-diagram-for-library-management-system/ (Accessed: 25 April 2025).