Library Extension Assignment

Table of Contents

1. Overview 2
2. Class Diagram 4
3. Object Diagram 4
4. Sequence Diagrams 5
5. Constraints 8
6. Testing 9
7. State Machine Diagrams 13
8. Source Code 18
9. Discussion and analysis 28

1. Overview

For the Software Engineering 2 assignment I chose the “Extend and test a more comprehensive USE model for the library system in USE” option. Here are the use cases I added.

1. Defective Use Case:

Description: The "Defective" use case handles the scenario where a member notices that a copy of a book is damaged or defective in some way, like torn pages or a broken spine. The member reports this issue to the library staff so that appropriate action can be taken.

Steps:

1. The member identifies a copy of a book that is damaged while browsing or borrowing it.
2. The member notifies the library staff about the damaged copy, providing details about the issue.
3. The library staff records the report and marks the copy as defective in the system.
4. The system updates the status of the copy to indicate that it's damaged, preventing further borrowing until it's repaired.
5. The system may impose a fine on the member for damaging the book, depending on library policies.
6. The damaged copy is set aside for repair, and the member may be offered a replacement copy if available.

Outcome: The damaged copy is flagged for repair, ensuring that it's not lent out to other members until it's restored to a good condition. The member may incur a fine for damaging the book, serving as a deterrent against mishandling library materials.

2. Repair Use Case:

Description: The "Repair" use case addresses the process of fixing damaged copies of books that have been reported as defective by members. This involves assessing the extent of damage, performing necessary repairs, and restoring the copy to a condition suitable for lending.

Steps:

1. Library staff identifies copies that have been marked as defective and need repair.
2. The damaged copies are collected and assessed to determine the nature and extent of damage.
3. Depending on the damage, appropriate repair actions are taken, such as mending torn pages, replacing covers, or fixing bindings.
4. Once repairs are completed, the copy's condition status is updated in the system to indicate that it's been restored to good condition.
5. The copy is returned to circulation, making it available for borrowing by library members.

Outcome: Damaged copies are restored to a usable condition, ensuring that library materials remain in good shape for patrons to borrow. This process helps maintain the quality and integrity of the library's collection.

3. Pay Penalty Use Case:

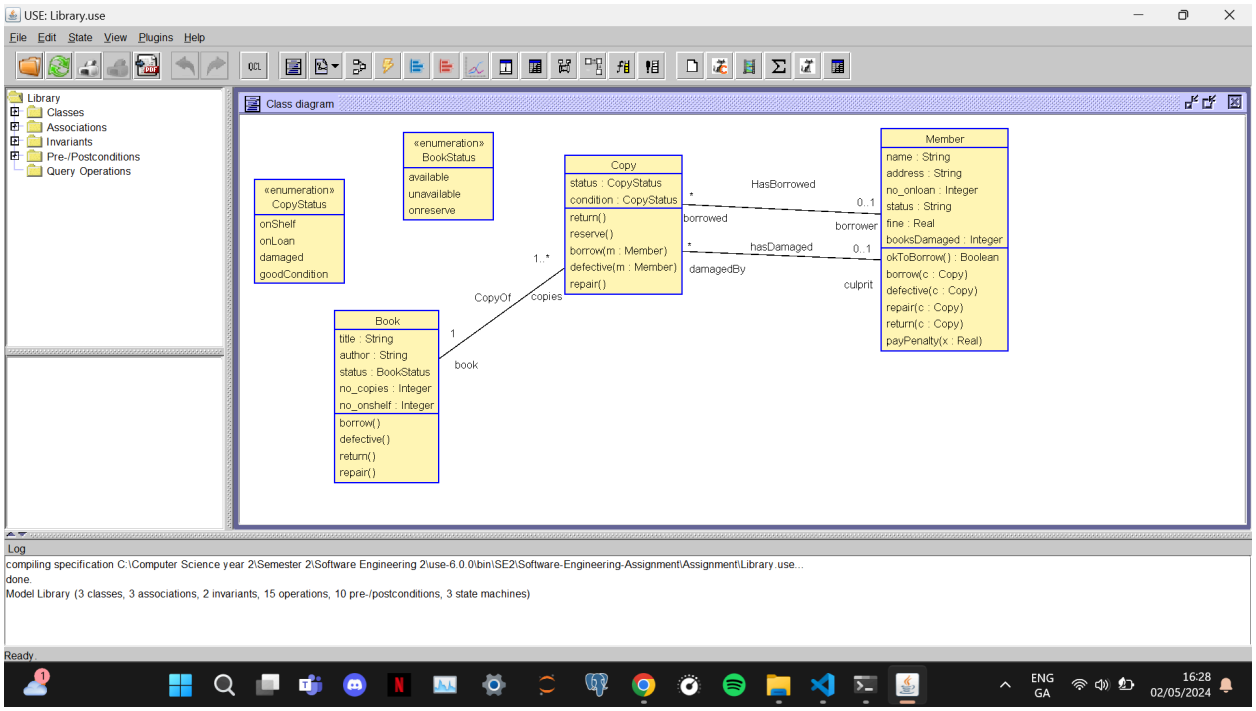
Description: The "Pay Penalty" use case deals with members settling fines or penalties incurred due to overdue items, damaged books, or other library policy violations. Members can make payments to clear their outstanding fines and regain good standing with the library.

Steps:

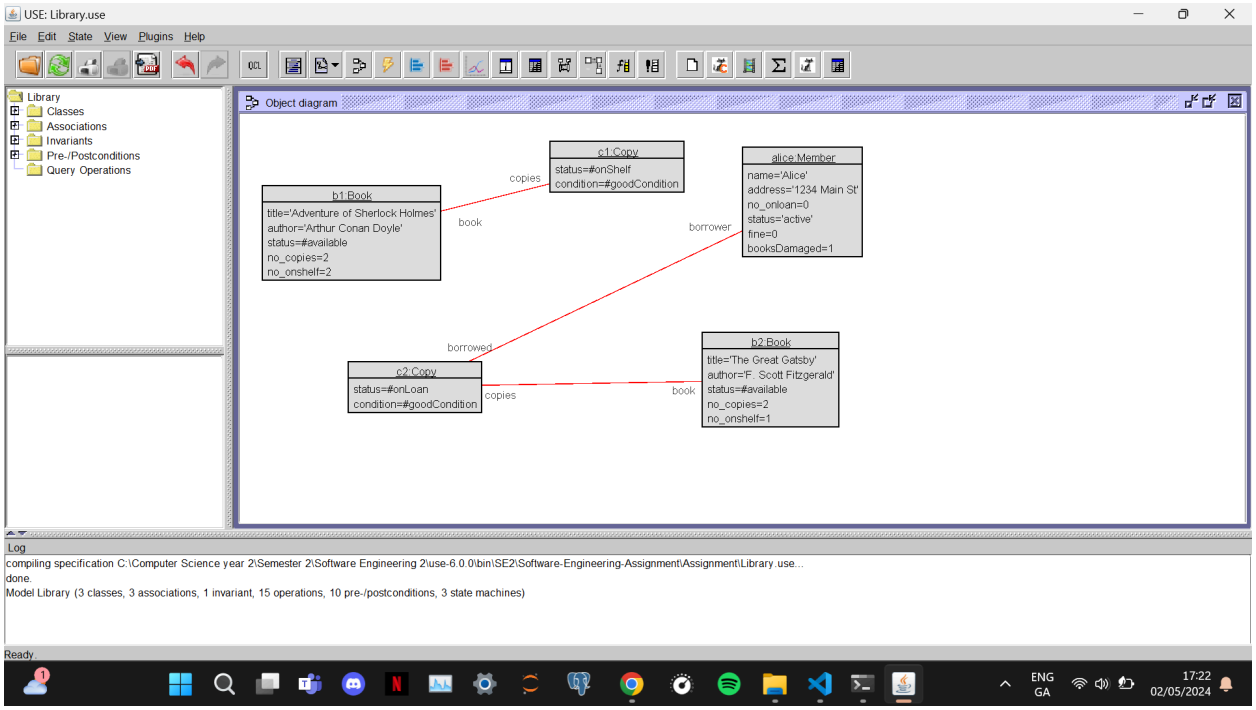
1. A member with an outstanding fine checks their account to view the amount owed.
2. The member decides to pay off the fine and approaches the library staff for payment.
3. The staff calculates the total amount owed and accepts payment from the member, either in cash, card, or through another accepted payment method.
4. The staff updates the member's account to reflect the payment and reduces the fine balance accordingly.
5. The member receives confirmation of the payment and may request a receipt for their records.

Outcome: By paying the penalty, the member clears their debt with the library and avoids any further consequences, such as restricted borrowing privileges or additional fines. The library benefits from the collection of fines as a means of recouping costs and reinforcing compliance with library policies.

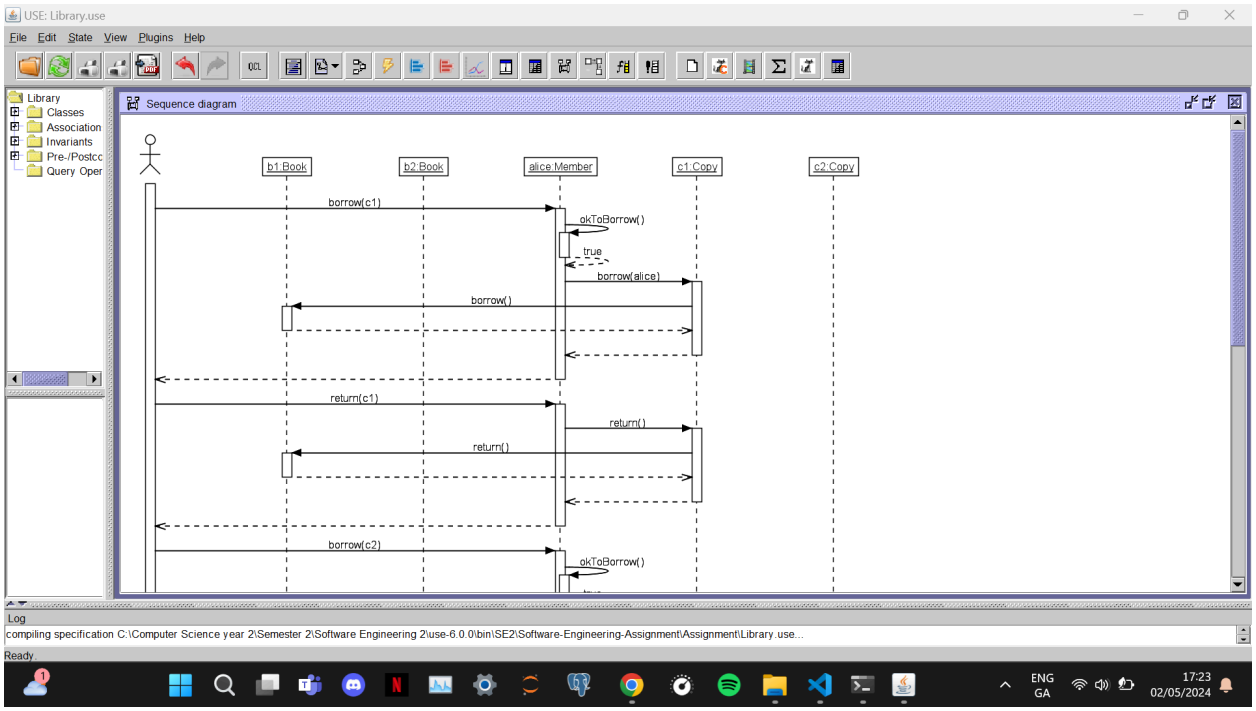
2. Class Diagram

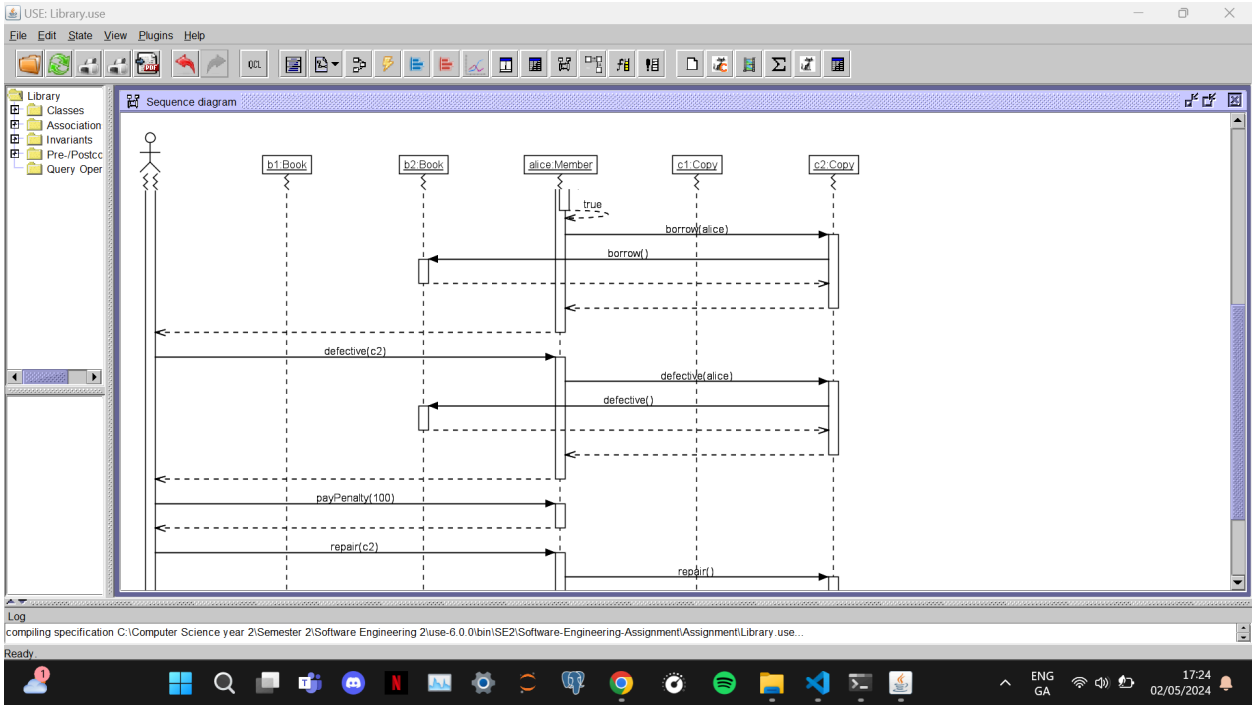


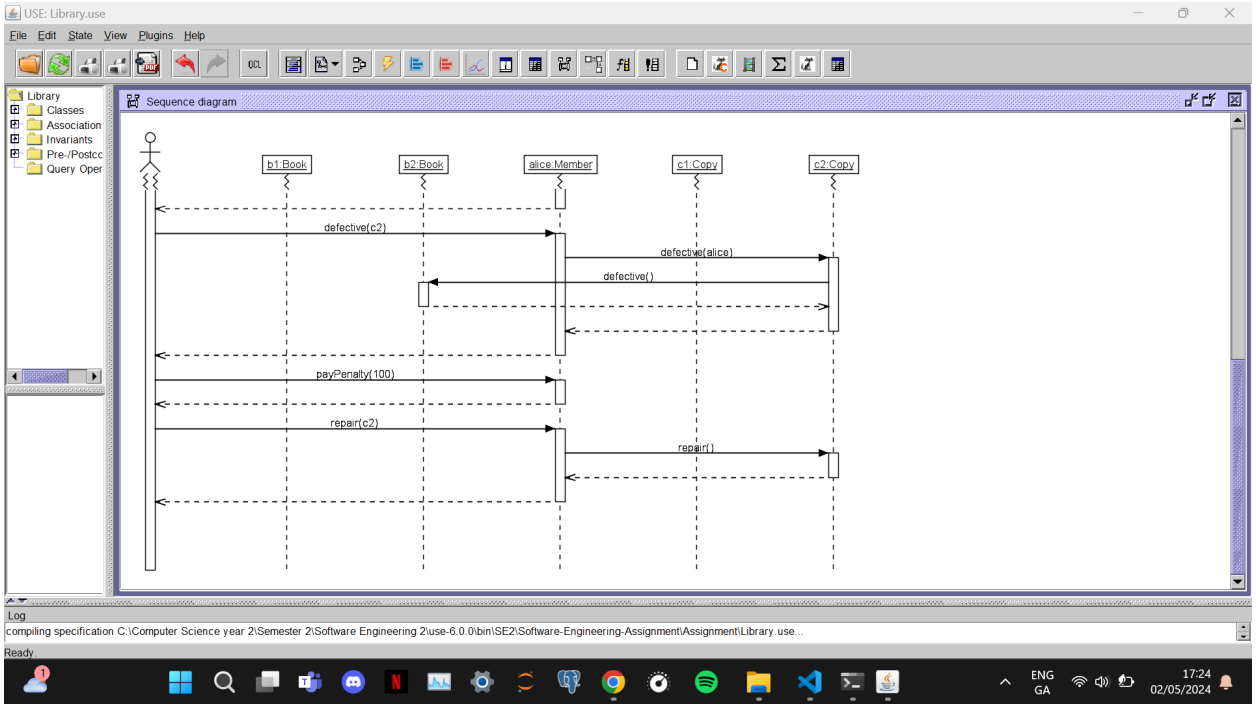
3. Object Diagram

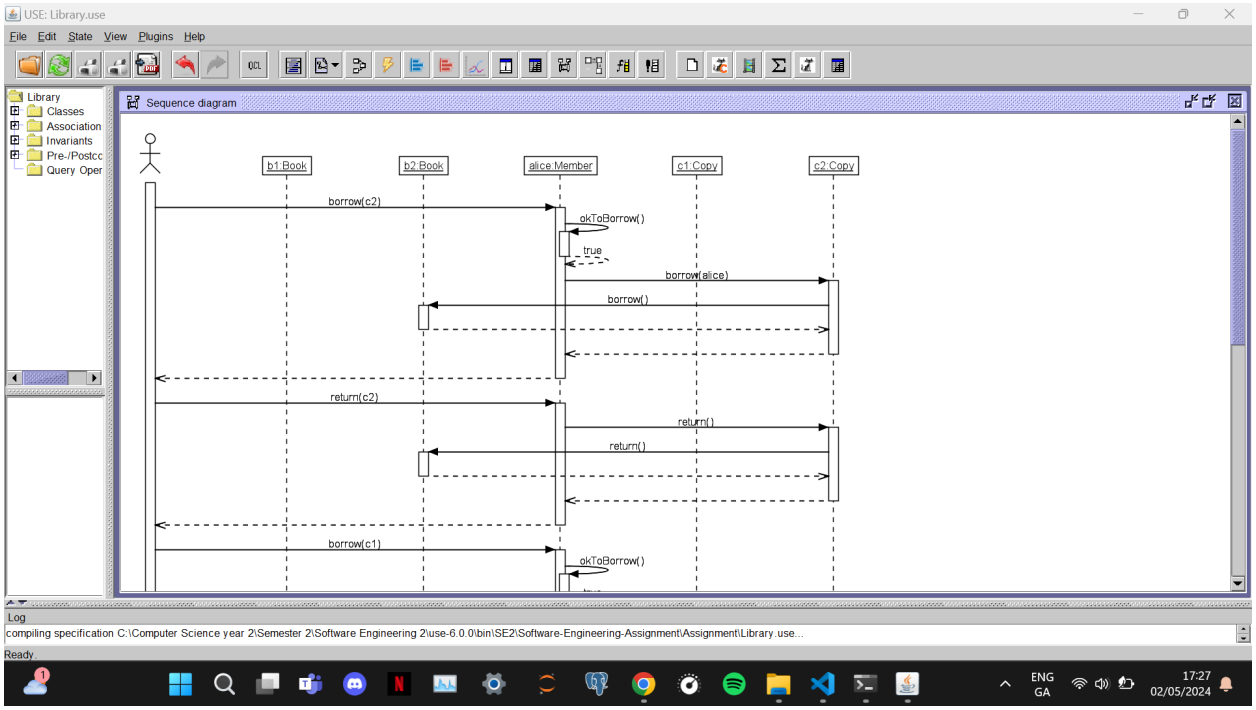


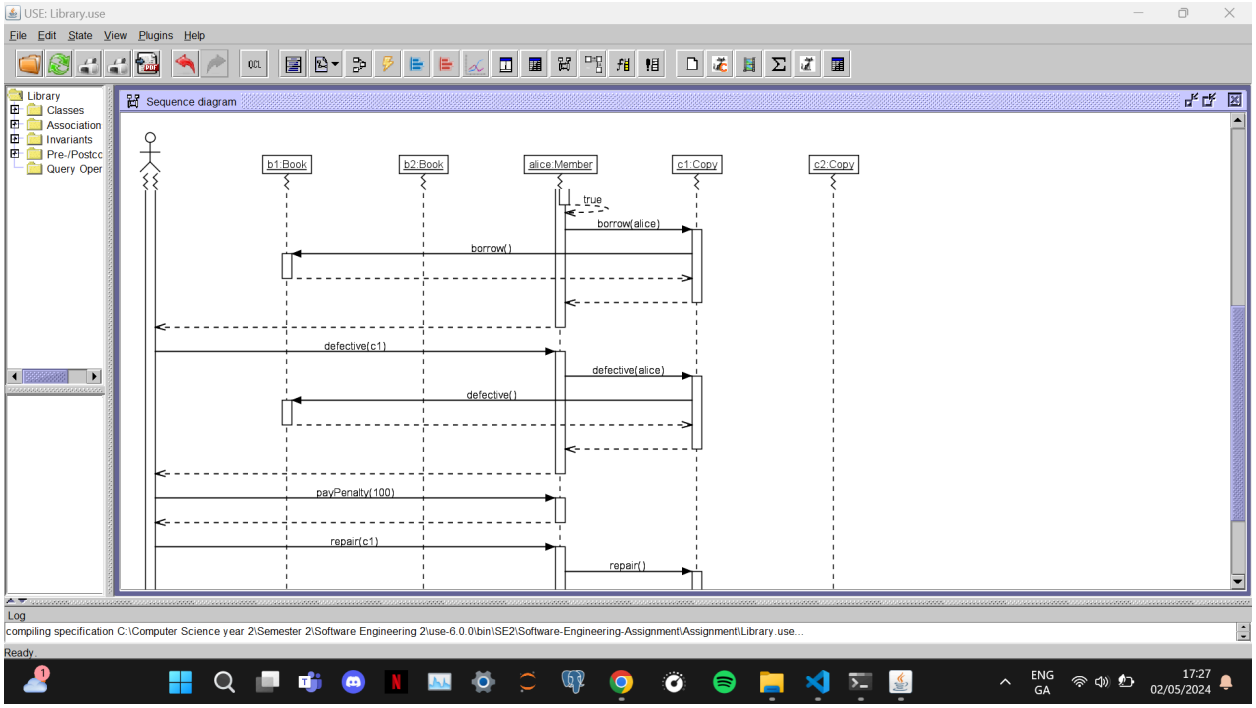
4. Sequence Diagrams

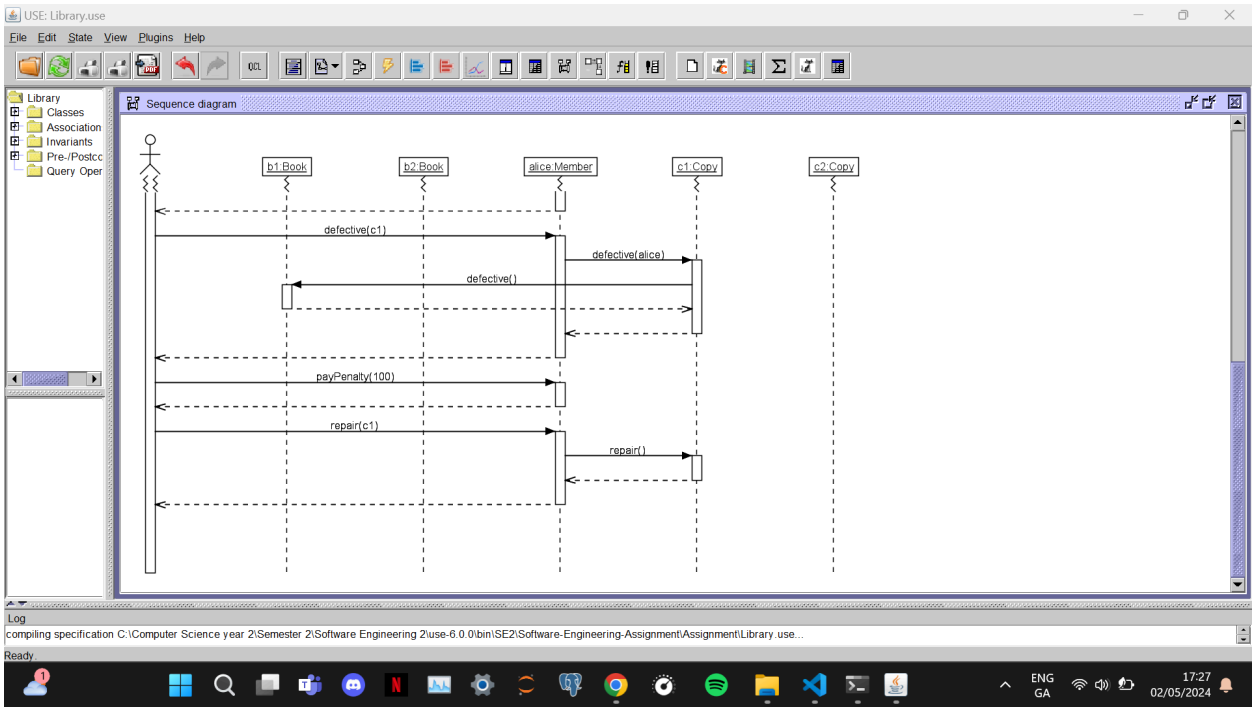





USE: Library.use
File Edit State View Plugins Help
Library
Classes
Association
Invariants
Pre-/Postcc
Query Oper
Sequence diagram
b1:Book
b2:Book
alice:Member
c1:Copy
c2:Copy
defective(c2)
defective(alice)
defective()
payPenalty(100)
repair(c2)
repair()
Log
compiling specification C:\Computer Science year 2\Semester 2\Software Engineering 2\use-6.0.0\bin\SE2\Software-Engineering-Assignment\Assignment\Library.use...
Ready.
ENG
GA
17:24
02/05/2024


USE: Library.use
File Edit State View Plugins Help
Library
Classes
Association
Invariants
Pre-/Postcc
Query Oper
Sequence diagram
b1:Book
b2:Book
alice:Member
c1:Copy
c2:Copy
borrow(c2)
okToBorrow()
true
borrow(alice)
borrow()
return(c2)
return()
return()
borrow(c1)
okToBorrow()
Log
compiling specification C:\Computer Science year 2\Semester 2\Software Engineering 2\use-6.0.0\bin\SE2\Software-Engineering-Assignment\Assignment\Library.use...
Ready.
ENG
GA
17:27
02/05/2024


USE: Library.use
File Edit State View Plugins Help
Library
Classes
Association
Invariants
Pre-/Postcc
Query Oper
Sequence diagram
b1:Book
b2:Book
alice:Member
c1:Copy
c2:Copy
true
borrow(alice)
borrow()
defective(c1)
defective(alice)
defective()
payPenalty(100)
repair(c1)
repair()
Log
compiling specification C:\Computer Science year 2\Semester 2\Software Engineering 2\use-6.0.0\bin\SE2\Software-Engineering-Assignment\Assignment\Library.use...
Ready.
ENG
GA
17:27
02/05/2024

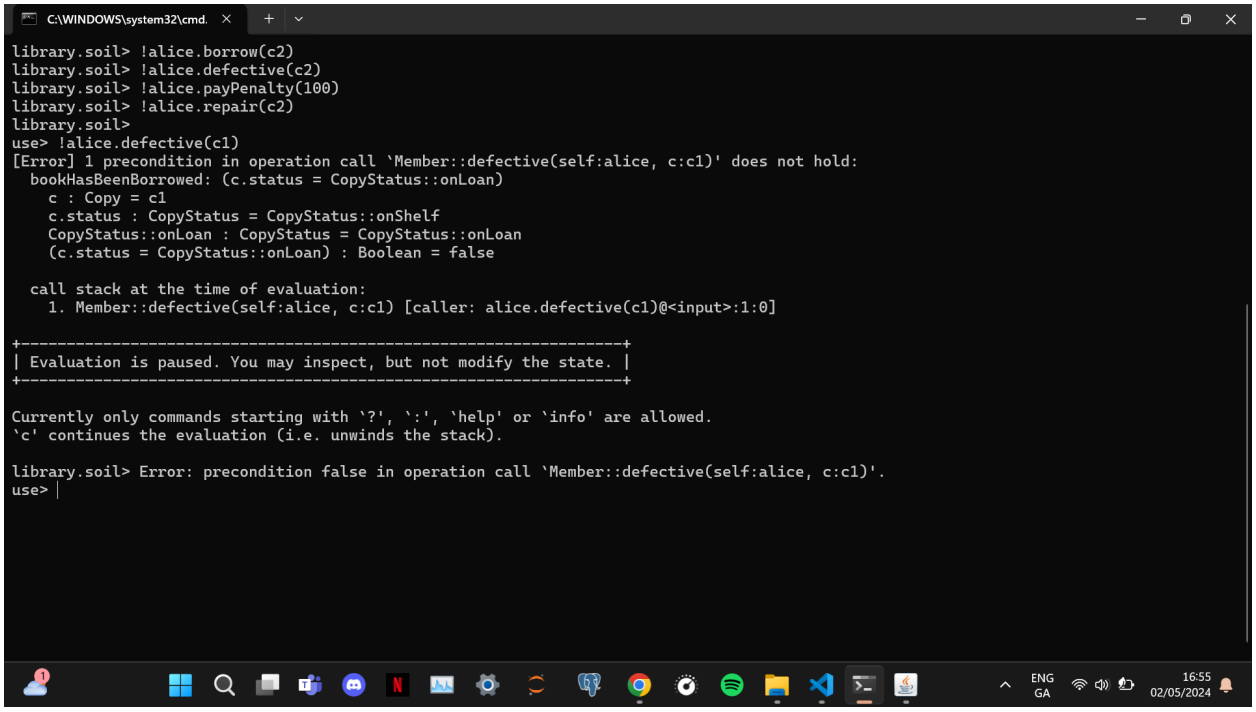

USE: Library.use
File Edit State View Plugins Help
Library
Classes
Association
Invariants
Pre-/Postcc
Query Oper
Sequence diagram
b1:Book
b2:Book
alice:Member
c1:Copy
c2:Copy
defective(c1)
defective(alice)
defective()
payPenalty(100)
repair(c1)
repair()
Log
compiling specification C:\Computer Science year 2\Semester 2\Software Engineering 2\use-6.0.0\bin\SE2\Software-Engineering-Assignment\Assignment\Library.use...
Ready.
ENG
GA
17:27
02/05/2024

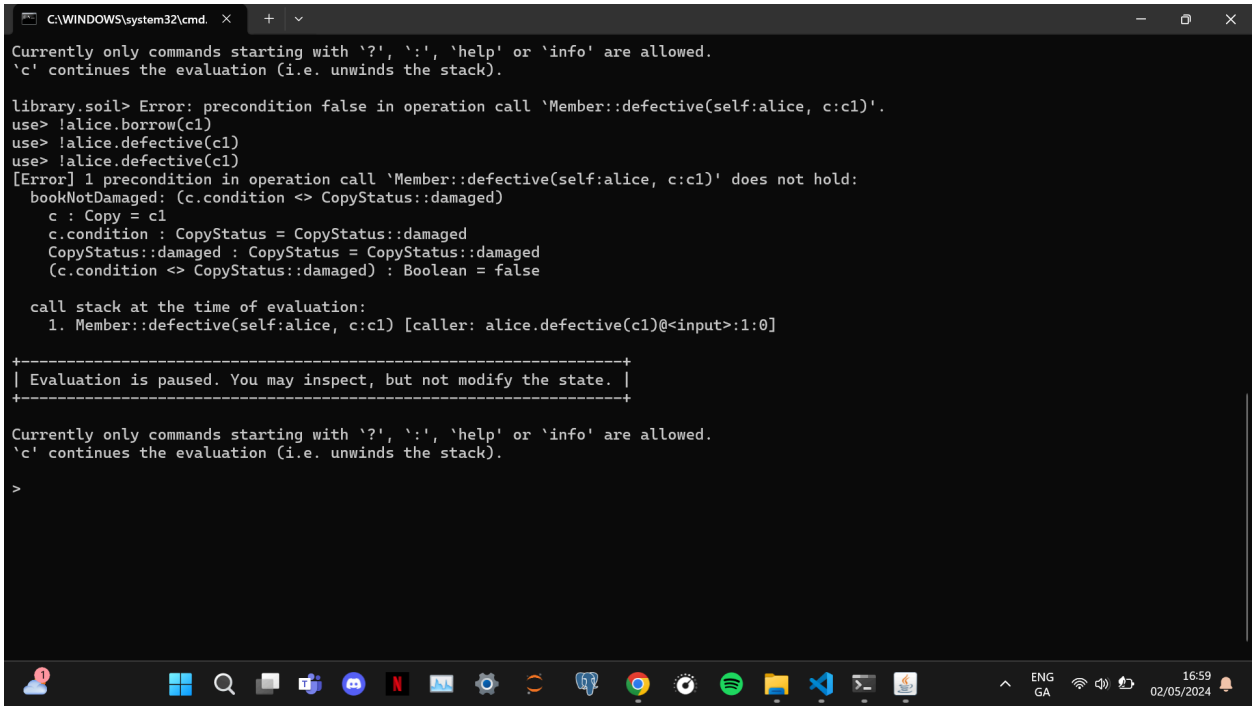
5. Constraints

```
-- Added this

-- Marking a Book as Defective

context Member::defective(c: Copy)

    -- Preconditions: Only a book currently on loan can be marked as defective

    pre bookHasBeenBorrowed: c.status = #onLoan


    -- Preconditions: Ensure you can't mark a book as defective if it's already damaged

    pre bookNotDamaged: c.condition <> #damaged


    -- Postcondition: After marking, the book's condition should be set to damaged

    post bookHasBeenDamaged: c.condition = #damaged


    -- Constraints: Books can only be marked as defective if they are either currently
borrowed or already marked as damaged


-- Paying a Penalty

context Member::payPenalty(x: Real)

    -- Preconditions: You can only pay a fine if you have an outstanding fine to settle

    pre hasToPayFine: self.fine > 0


    -- Preconditions: The amount being paid must be greater than zero

    pre BiggerThanZero: x > 0


    -- Postcondition: Paying a fine shouldn't result in a negative fine balance

    post NonNegativeFine: self.fine >= 0
```

6. Testing

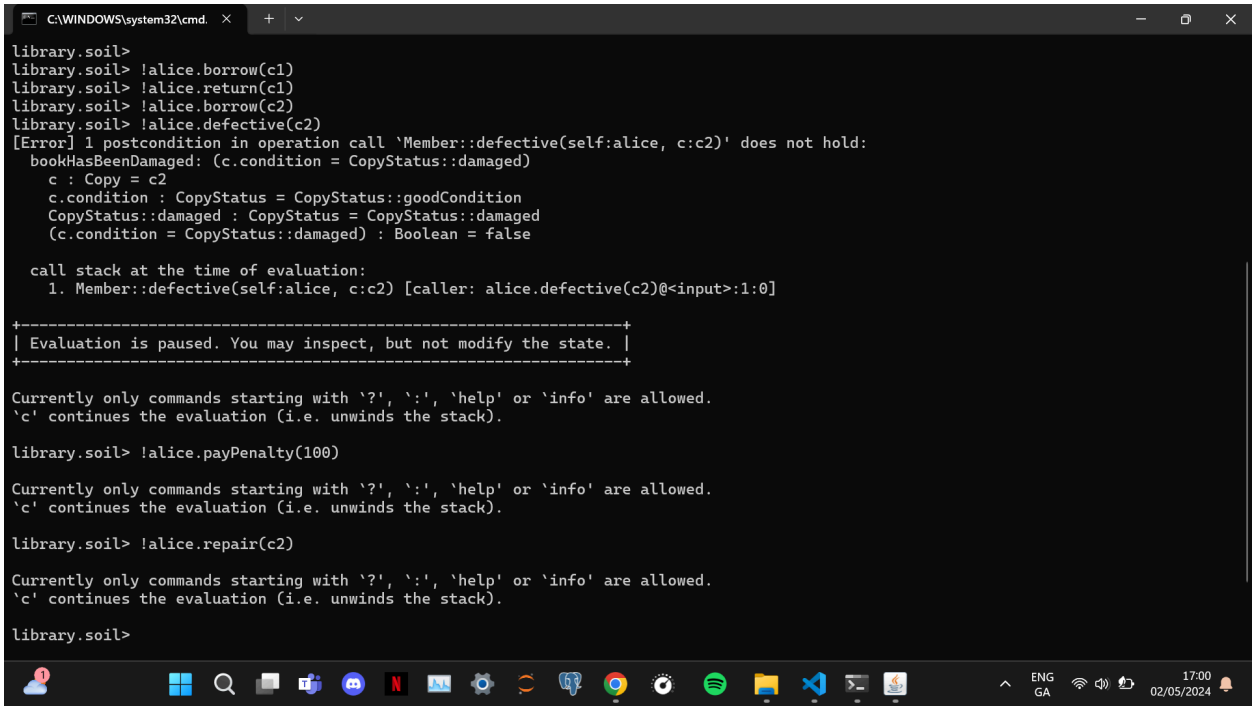
So for testing I got every pre and post condition I made to fail to make sure that my software catches any errors, whether its logic or input errors. I do all this in the order you see above in the constraints section so I test the precondition BookHasBeenBorrowed first. I've also left the testing lines commented out if you want to experiment with them in the soil file for final submission. I'd also like to add that it's impossible to test this software with both soil and OCL. My implementations are in soil as the assignment told me that's the way we need to do it.

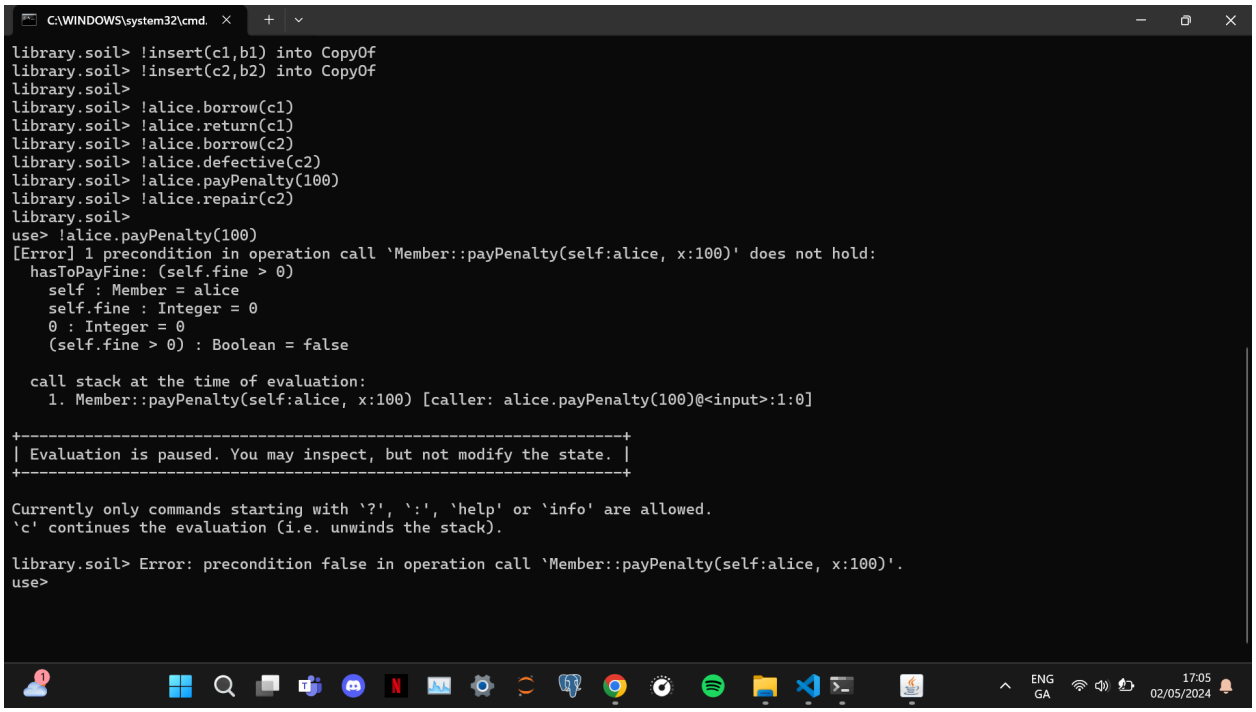


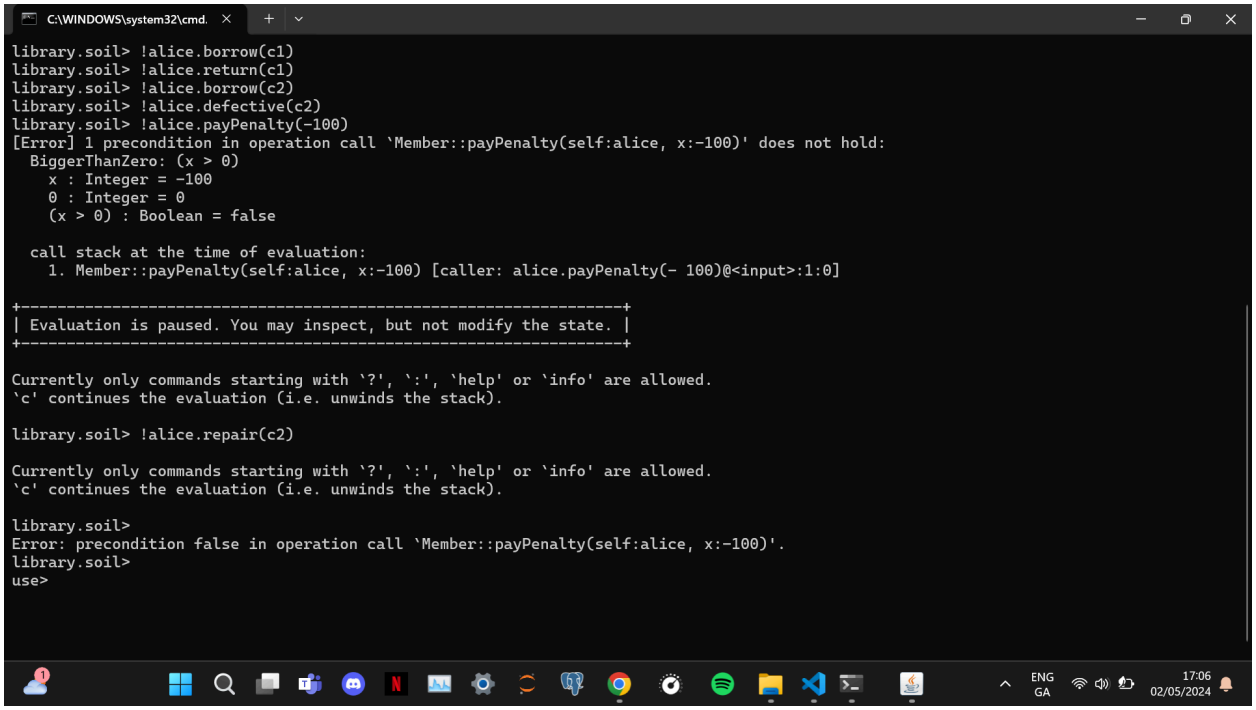


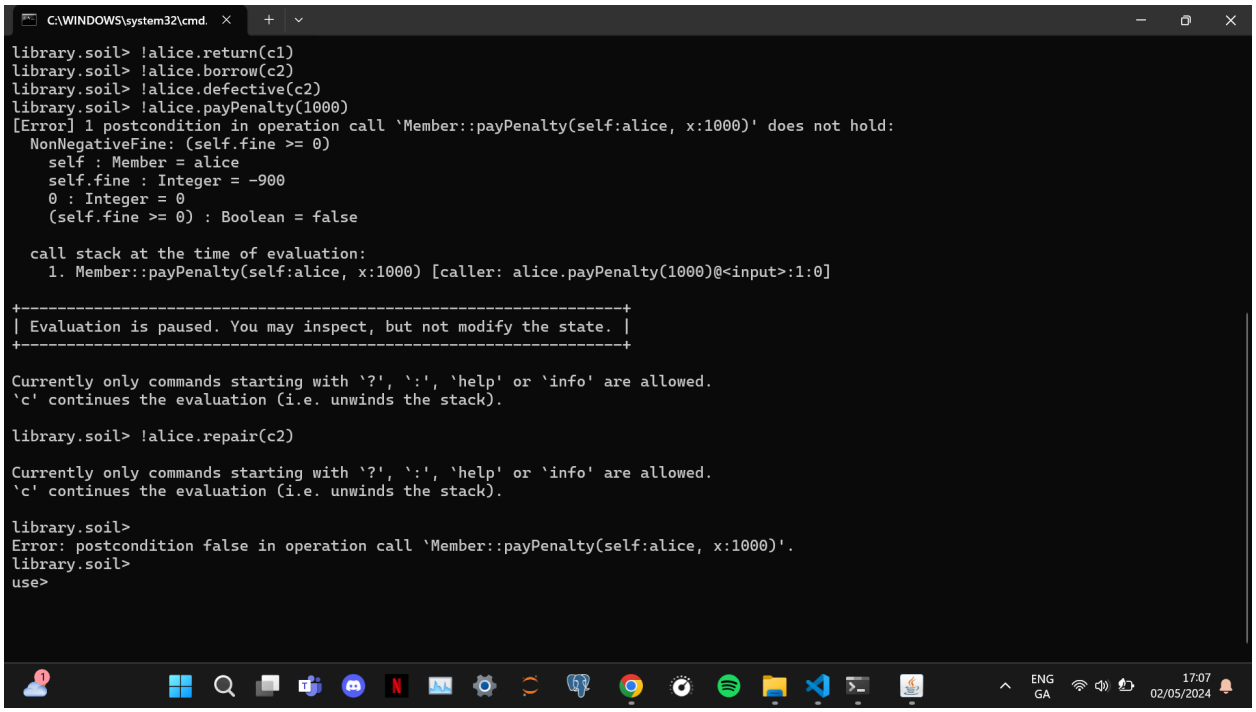
So the reason this next postcondition fails is because I purposely code the condition of the copy to not change so this picks it up. So the book is marked as defective but the condition isn't updated. I temporarily changed this, for the purpose of testing so this is not the final submitted code.

```
        -- Marks the copy as defective and notifies the associated book

        defective(m : Member)

        begin

            -- self.condition := #damaged;

            self.book.defective()

        end
```


C:\WINDOWS\system32\cmd.
library.soil>
library.soil> !alice.borrow(c1)
library.soil> !alice.return(c1)
library.soil> !alice.borrow(c2)
library.soil> !alice.defective(c2)
[Error] 1 postcondition in operation call `Member::defective(self:alice, c:c2)' does not hold:
bookHasBeenDamaged: (c.condition = CopyStatus::damaged)
c : Copy = c2
c.condition : CopyStatus = CopyStatus::goodCondition
CopyStatus::damaged : CopyStatus = CopyStatus::damaged
(c.condition = CopyStatus::damaged) : Boolean = false
call stack at the time of evaluation:
1. Member::defective(self:alice, c:c2) [caller: alice.defective(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).
library.soil> !alice.payPenalty(100)
Currently only commands starting with `?', `:', `help' or `info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).
library.soil> !alice.repair(c2)
Currently only commands starting with `?', `:', `help' or `info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).
library.soil>
ENG
GA
17:00
02/05/2024

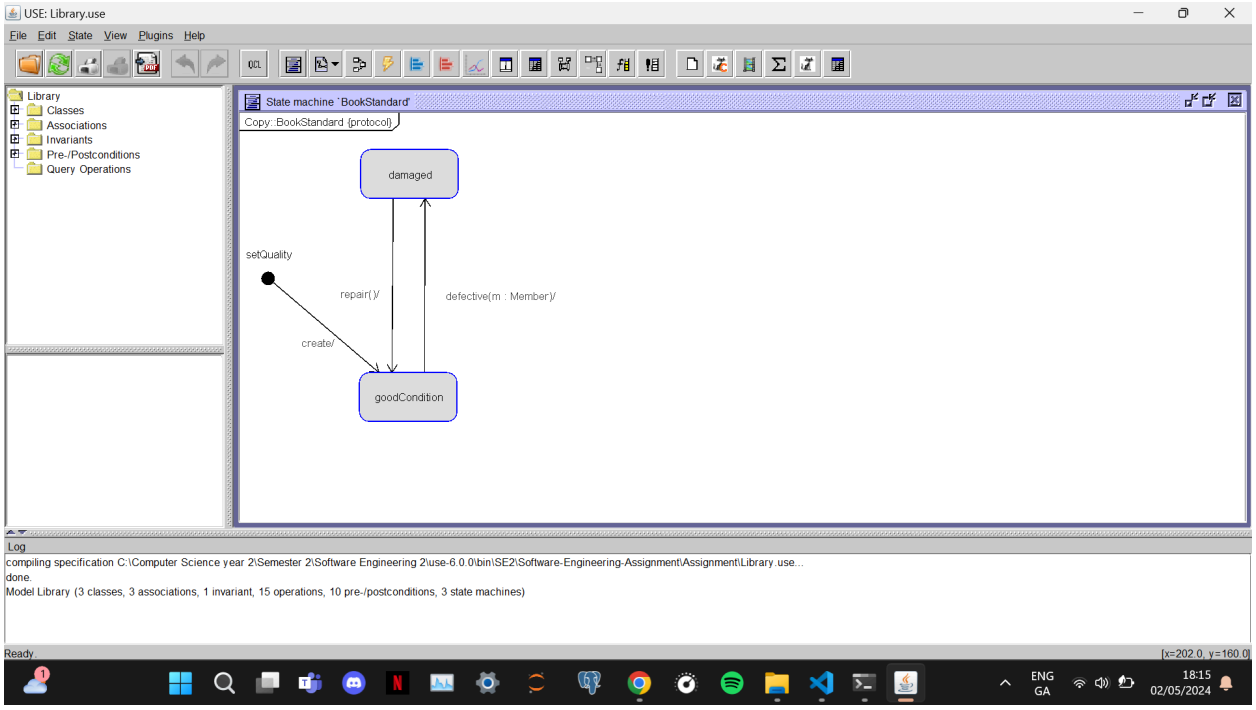

C:\WINDOWS\system32\cmd.
library.soil> !insert(c1,b1) into CopyOf
library.soil> !insert(c2,b2) into CopyOf
library.soil>
library.soil> !alice.borrow(c1)
library.soil> !alice.return(c1)
library.soil> !alice.borrow(c2)
library.soil> !alice.defective(c2)
library.soil> !alice.payPenalty(100)
library.soil> !alice.repair(c2)
library.soil>
use> !alice.payPenalty(100)
[Error] 1 precondition in operation call `Member::payPenalty(self:alice, x:100)' does not hold:
hasToPayFine: (self.fine > 0)
self : Member = alice
self.fine : Integer = 0
0 : Integer = 0
(self.fine > 0) : Boolean = false
call stack at the time of evaluation:
1. Member::payPenalty(self:alice, x:100) [caller: alice.payPenalty(100)@<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: precondition false in operation call `Member::payPenalty(self:alice, x:100)'.
use>
ENG
GA
17:05
02/05/2024


C:\WINDOWS\system32\cmd.
library.soil> !alice.borrow(c1)
library.soil> !alice.return(c1)
library.soil> !alice.borrow(c2)
library.soil> !alice.defective(c2)
library.soil> !alice.payPenalty(-100)
[Error] 1 precondition in operation call `Member::payPenalty(self:alice, x:-100)' does not hold:
BiggerThanZero: (x > 0)
x : Integer = -100
0 : Integer = 0
(x > 0) : Boolean = false
call stack at the time of evaluation:
1. Member::payPenalty(self:alice, x:-100) [caller: alice.payPenalty(- 100)@<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> !alice.repair(c2)
Currently only commands starting with `?', `:', `help' or `info' are allowed.
`c' continues the evaluation (i.e. unwinds the stack).
library.soil>
Error: precondition false in operation call `Member::payPenalty(self:alice, x:-100)'.
library.soil>
use>
ENG
GA
17:06
02/05/2024

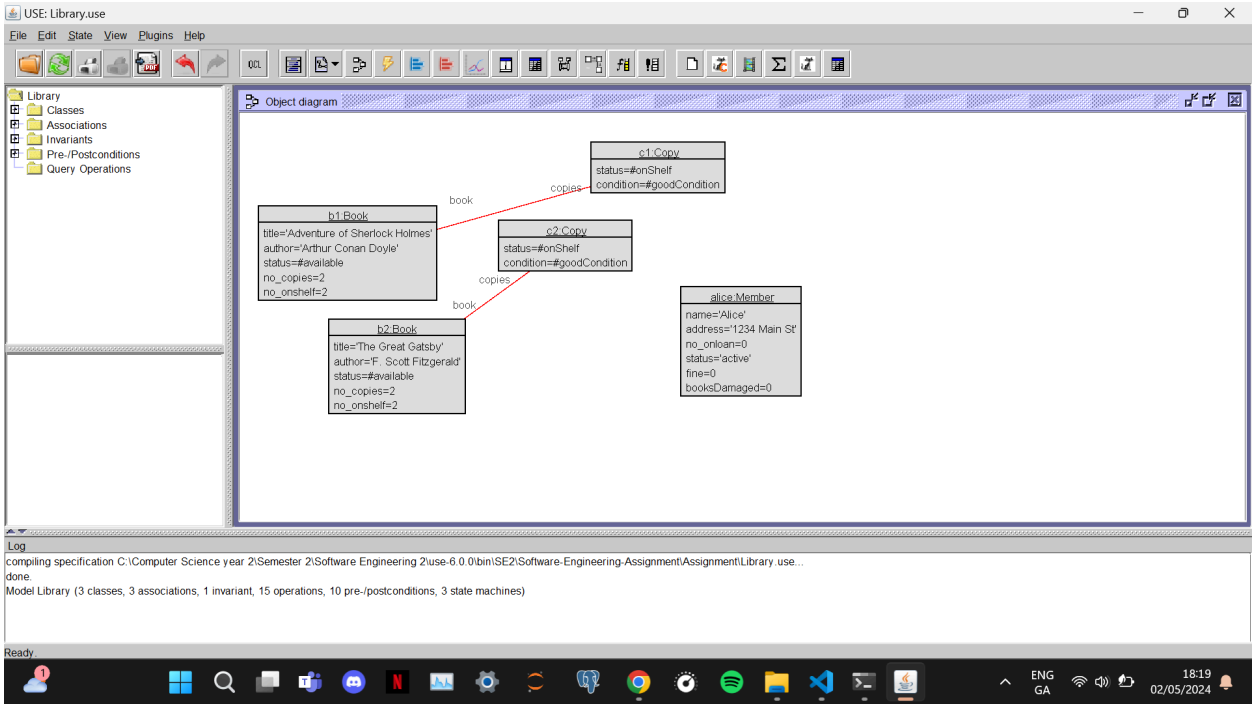

C:\WINDOWS\system32\cmd.
library.soil> !alice.return(c1)
library.soil> !alice.borrow(c2)
library.soil> !alice.defective(c2)
library.soil> !alice.payPenalty(1000)
[Error] 1 postcondition in operation call `Member::payPenalty(self:alice, x:1000)' does not hold:
NonNegativeFine: (self.fine >= 0)
self : Member = alice
self.fine : Integer = -900
0 : Integer = 0
(self.fine >= 0) : Boolean = false
call stack at the time of evaluation:
1. Member::payPenalty(self:alice, x:1000) [caller: alice.payPenalty(1000)@<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> !alice.repair(c2)
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::payPenalty(self:alice, x:1000)'.
library.soil>
use>
ENG
GA
17:07
02/05/2024

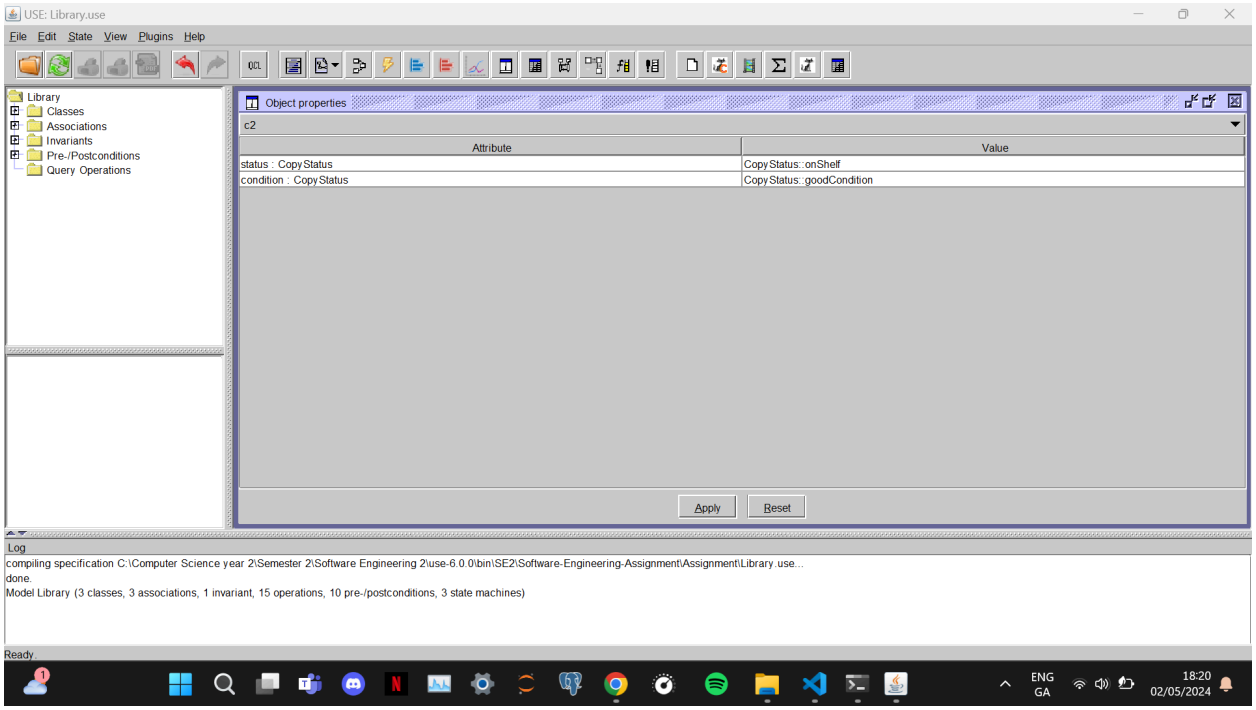
7. State Machine Diagrams

So the state machine I built was for the copy class looking at a book's condition over time. Keeps track of the states the copy of the book is in. Here are all the various states the copy of a book can be in when its borrowed, made defective, when a fine is paid and when the copy is repaired and returned.

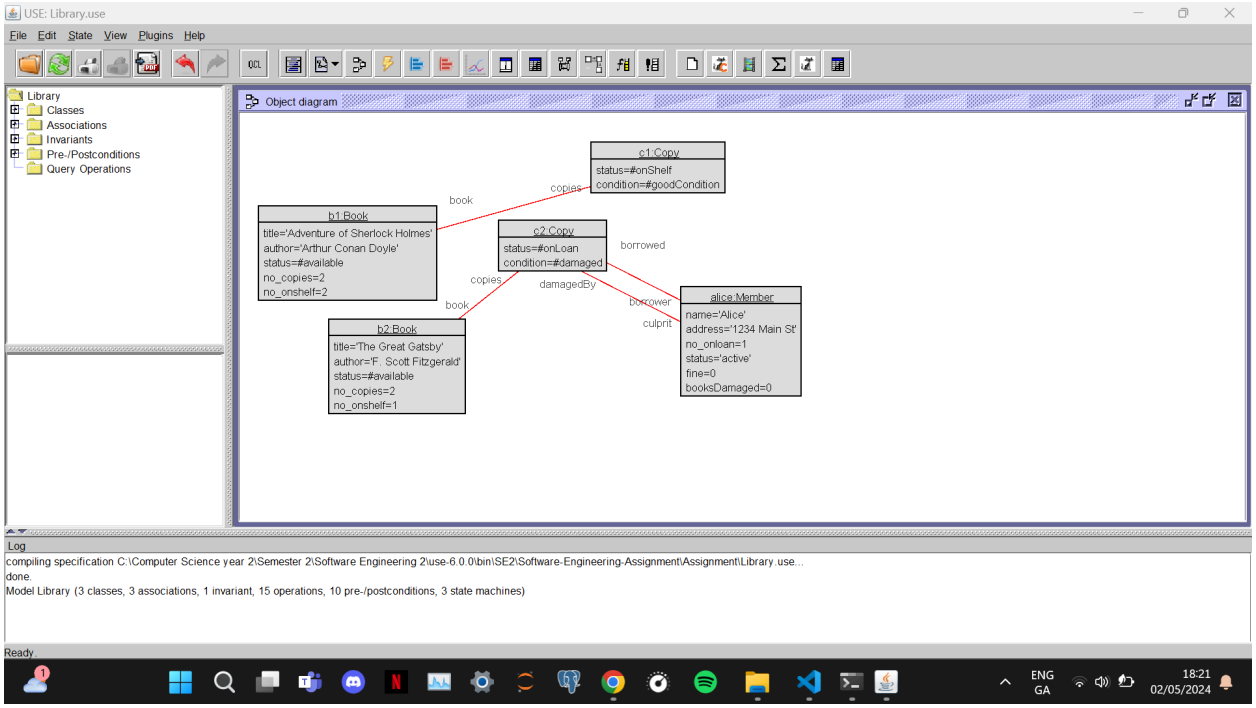


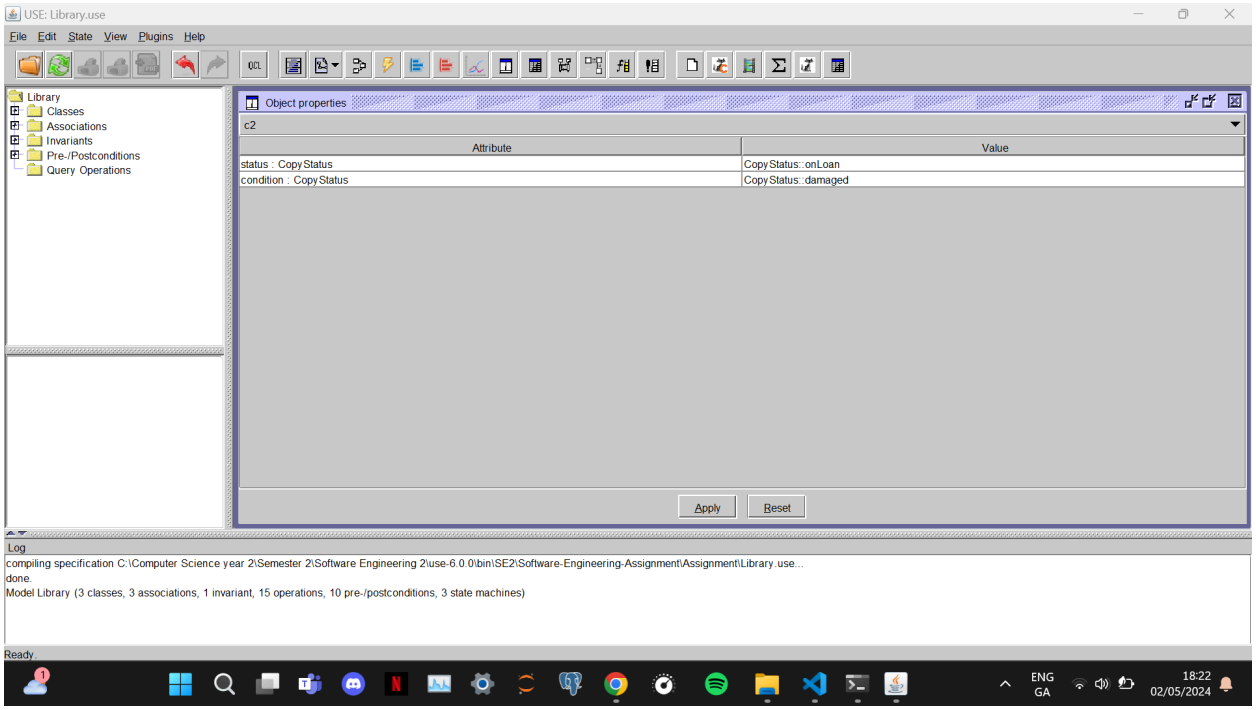
This is before changing any states. I'm using a copy(c2) of a book as an example.



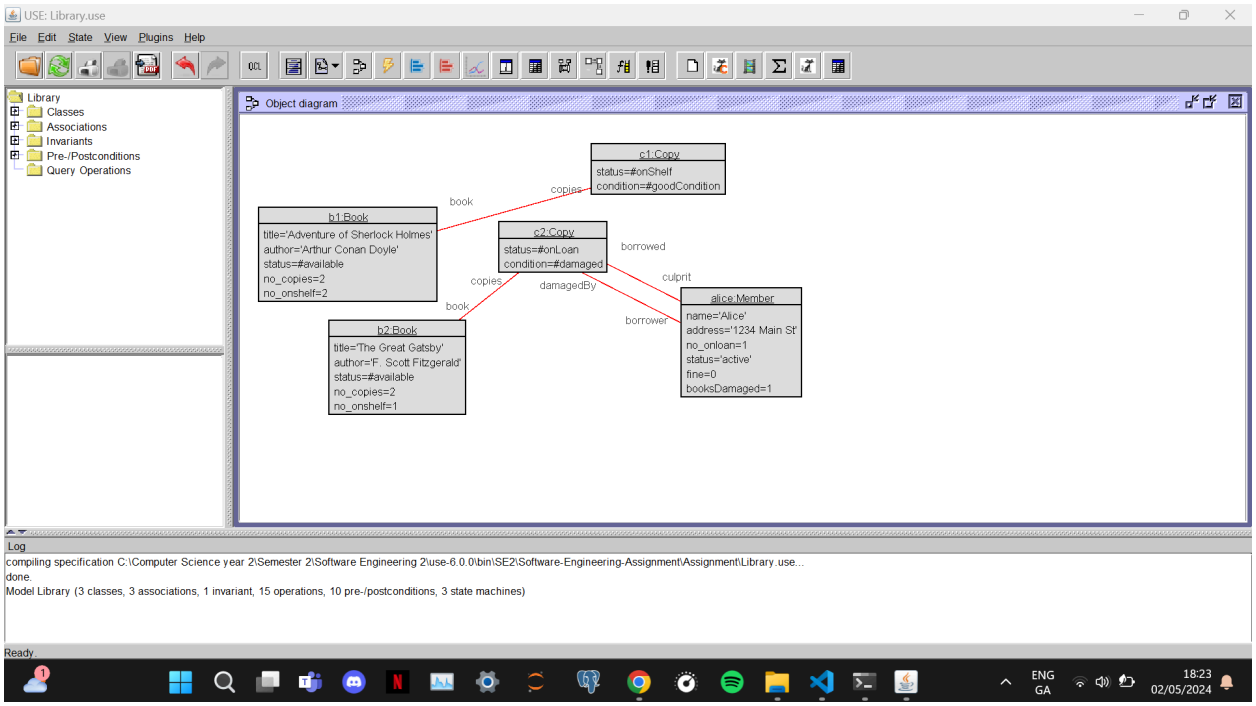


Now the book has been damaged and reported by Alice. You can clearly see the change in states. Now it's on loan and damaged. An association has also been created between Alice and the copy of the book. She is responsible for the damages.

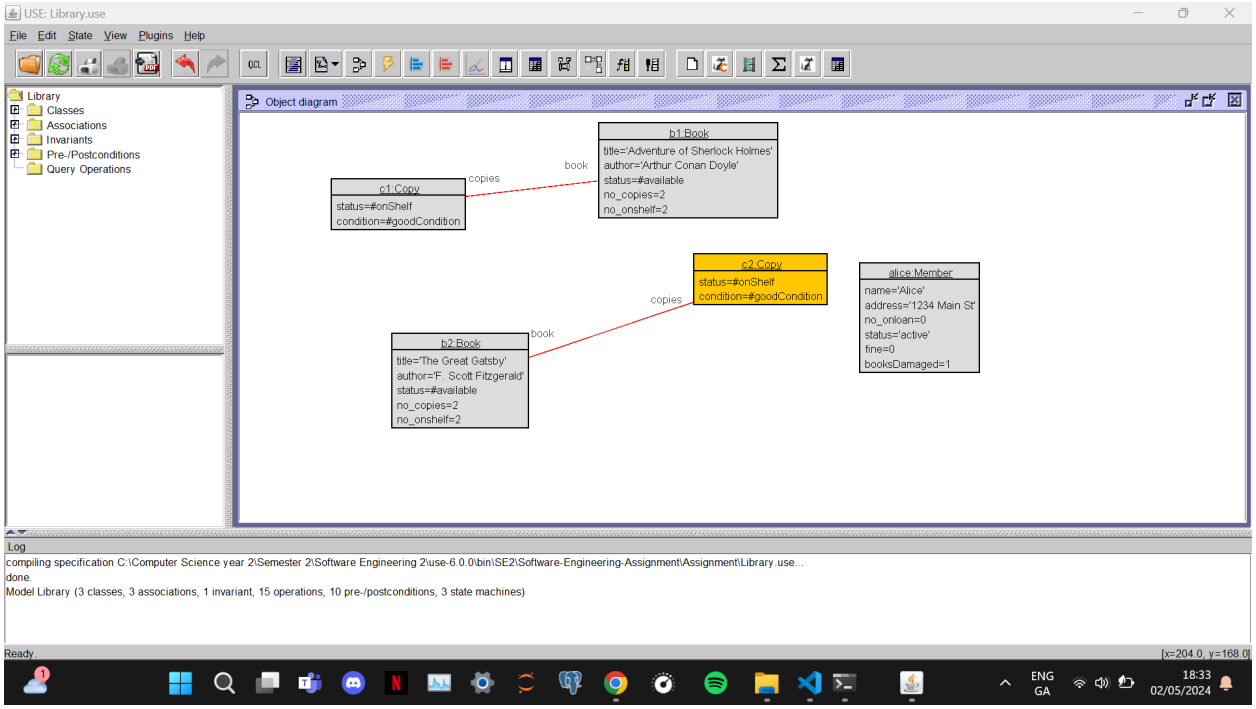


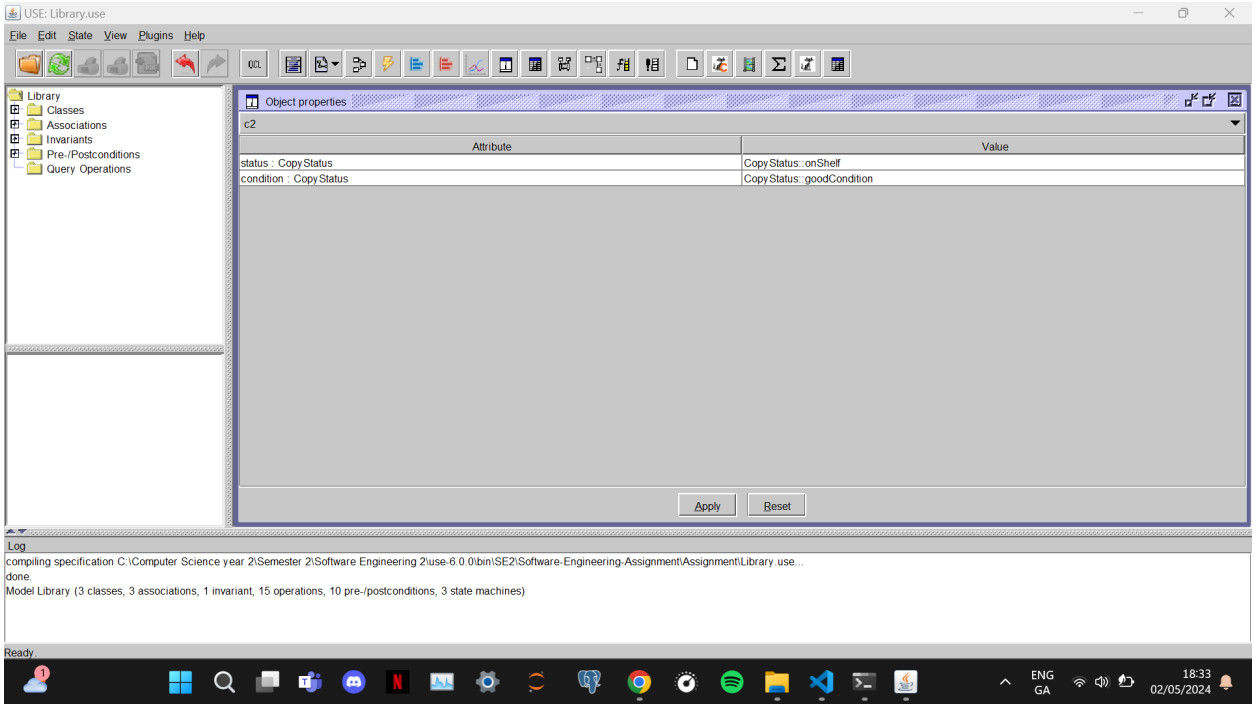


Alice then pays the fine and the amount of books she has damaged has gone up by one.



The book is then repaired. The copy goes back on the shelf simultaneously. Back in good condition.




USE: Library.use
File Edit State View Plugins Help
Library
Classes
Associations
Invariants
Pre-/Postconditions
Query Operations
Object properties
c2
Attribute
Value
status : CopyStatus
CopyStatus::onShelf
condition : CopyStatus
CopyStatus::goodCondition
Apply
Reset
Log
compiling specification C:\Computer Science year 2\Semester 2\Software Engineering 2\use-6.0.0\bin\SE2\Software-Engineering-Assignment\Assignment\Library.use...
done.
Model Library (3 classes, 3 associations, 1 invariant, 15 operations, 10 pre-/postconditions, 3 state machines)
Ready.
ENG
GA
18:33
02/05/2024

8. Source Code

Library.use

```
model Library


enum BookStatus { available, unavailable, 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


        -- Added this

        defective()

        begin

        end


        return()
```

```
        begin
            self.no_onshelf := self.no_onshelf + 1;
            self.status := #available
        end
        post: no_onshelf = no_onshelf@pre + 1

        -- Added this
        repair()
        begin
        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

enum CopyStatus {onShelf, onLoan, damaged, goodCondition}

class Copy
    attributes
        status : CopyStatus init = #onShelf;
```

```
    -- Added this
    -- Represents the condition of a library copy.
    condition : CopyStatus init = #goodCondition
operations
    return()
    begin
        self.status := #onShelf;
        self.book.return()
    end

    reserve()

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

    -- Added this
    -- Marks the copy as defective and notifies the associated book
    defective(m : Member)
    begin
        self.condition := #damaged;
        self.book.defective()
    end

    -- Added this
    -- Restores the copy's condition to goodCondition
    repair()
    begin
        self.condition := #goodCondition
```

```
        end


    statemachines
        psm States
        states
            newCopy : initial
            onShelf
            onLoan
        transitions
            newCopy -> onShelf {create}
            onShelf -> onLoan { borrow() }
            onLoan -> onShelf { return() }
        end
        -- Added this
        -- Represents a state machine for managing the quality of a book.
        psm BookStandard
        states
            -- Initial state when a book's quality is being set
            setQuality : initial


            -- Represents a book in good condition
            goodCondition


            -- Represents a book that has been damaged
            damaged


        transitions
            -- Transition from setting the quality to a state of good condition
            setQuality -> goodCondition { create }


            -- Transition from good condition to damaged state, triggered when the book is
marked as defective
```

```
            goodCondition -> damaged { defective() }


            -- Transition from damaged state back to good condition after the book has been
repaired
            damaged -> goodCondition { repair() }
        end
end


class Member
  attributes
    name : String
    address : String
    no_onloan : Integer init = 0
    -- Represents the status of the member (e.g., active, suspended)
    status : String


    -- Represents the amount of fine the member has to pay
    fine : Real init = 0


    -- Represents the number of books damaged by the member
    booksDamaged : Integer init = 0
  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


    -- Added this
    -- Marks a copy of a book as defective
    defective(c: Copy)
    begin
        -- Insert the pair (self, c) into the collection of copies marked as damaged
        insert (self, c) into hasDamaged;


        -- Call the defective method on the copy, indicating it's marked as defective by
this member
        c.defective(self);


        -- Increase the member's fine by 100 for marking the book as defective
        self.fine := self.fine + 100;


        -- Increase the count of books damaged by this member
        self.booksDamaged := self.booksDamaged + 1
    end


    -- Added this: Repairs a copy of a previously damaged book
    repair(c: Copy)
    begin
        -- Remove the pair (self, c) from the collection of copies marked as damaged
```

```
        delete (self, c) from hasDamaged;


        -- Decrease the count of copies currently on loan by 1, indicating one less damaged
copy is being borrowed
        delete (self, c) from HasBorrowed;
        self.no_onloan := self.no_onloan - 1;


        -- Call the repair method on the copy, indicating it has been repaired
        c.repair();
        -- Call the reutrn method on the copy, indicating it has been returned
        c.return()
    end



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


    -- Represents the action of a member paying a penalty/fine
    payPenalty(x: Real)
    begin
        -- Decreases the member's fine by the amount specified (x)
        self.fine := self.fine - x
    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


-- Added this
-- Represents an association between members and copies, indicating which member marked
which copies as damaged
association hasDamaged between
    -- Specifies the cardinality and roles involved in the association:
    -- Member: 0 to 1 (optional) culprit role (the member who marked the copy as damaged)
    -- Copy: zero to many damagedBy role (the copies marked as damaged by the member)
    Member[0..1] role culprit
    Copy[*] role damagedBy
end


constraints


context Member::borrow(c:Copy)
    pre limit: self.no_onloan < 2
    pre cond1: self.borrowed->excludes(c)
    post cond2: self.borrowed->includes(c)


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


-- Added this
-- Marking a Book as Defective
context Member::defective(c: Copy)
    -- Preconditions: Only a book currently on loan can be marked as defective
```

```
    pre bookHasBeenBorrowed: c.status = #onLoan


    -- Preconditions: Ensure you can't mark a book as defective if it's already damaged
    pre bookNotDamaged: c.condition <> #damaged


    -- Postcondition: After marking, the book's condition should be set to damaged
    post bookHasBeenDamaged: c.condition = #damaged


    -- Constraints: Books can only be marked as defective if they are either currently
borrowed or already marked as damaged


-- Paying a Penalty
context Member::payPenalty(x: Real)
    -- Preconditions: You can only pay a fine if you have an outstanding fine to settle
    pre hasToPayFine: self.fine > 0


    -- Preconditions: The amount being paid must be greater than zero
    pre BiggerThanZero: x > 0


    -- Postcondition: Paying a fine shouldn't result in a negative fine balance
    post NonNegativeFine: self.fine >= 0
```

Library.soil

```
-- Script generated by USE 4.1.1


!new Book('b1')
!b1.title := 'Adventure of Sherlock Holmes'
!b1.author := 'Arthur Conan Doyle'


!new Book('b2')
```

```
!b2.title := 'The Great Gatsby'
!b2.author := 'F. Scott Fitzgerald'


!new Member('alice')
!alice.name := 'Alice'
!alice.address := '1234 Main St'
!alice.status := 'active'


!new Copy('c1')
!new Copy('c2')
!insert(c1,b1) into CopyOf
!insert(c2,b2) into CopyOf


!alice.borrow(c1)
!alice.return(c1)
-- For testing state machine also
!alice.borrow(c2)
!alice.defective(c2)
!alice.payPenalty(100)
-- !alice.payPenalty(-100) precondition BiggerThanZero
-- !alice.payPenalty(1000) postconditon NonNegativeFine
!alice.repair(c2)
-- !alice.defective(c1) precondition BookHasBeenBorrowed


-- !alice.defective(c1) precondition BookNotDamaged
-- !alice.defective(c1) precondition BookNotDamaged
-- !alice.payPenalty(100) precondition hasToPayFine
```

9. Discussion and analysis

Discussion and Analysis

The extension and testing of the Unified Software Engineering (USE) model for the library system introduces significant enhancements aimed at improving the management of library resources and member interactions. This discussion provides an analysis of the added functionality, along with insights into the testing approach and implementation details.

Added Use Cases:

Three new use cases were incorporated into the existing library system model:

1. Defective Use Case: This use case addresses the scenario where a member identifies a damaged copy of a book and reports it to library staff. The system marks the copy as defective, initiates repair procedures, and may impose fines on the member. This ensures that damaged copies are not circulated to other patrons and encourages responsible handling of library materials.

2. Repair Use Case: The Repair use case outlines the process of fixing damaged copies reported by members. Library staff assess the damage, perform necessary repairs, and update the copy's status in the system. By restoring damaged copies to good condition, the library maintains the quality of its collection and ensures availability for borrowing.

3. Pay Penalty Use Case: Members can settle fines or penalties incurred due to overdue items, damaged books, or policy violations. The system calculates the amount owed, accepts payments, and updates member accounts accordingly. This facilitates the resolution of outstanding fines, promoting compliance with library policies and maintaining positive member relations.

Implementation Details:

The implementation extends the existing class diagram with additional attributes, operations, and state machines to support the new use cases. Notable additions include the introduction of `defective`, `repair`, and `payPenalty` operations in the `Book` and `Member` classes, along with associated state transitions to manage book conditions and member fines.

Testing Approach:

A comprehensive testing strategy was adopted to validate the implemented functionalities. Each pre and post-condition defined in the constraints section was deliberately manipulated to ensure that the software accurately detects errors and enforces constraints. Testing was conducted sequentially, following the order specified in the constraints section, with testing lines provided for reference in the source code.

Discussion:

The extension of the USE model enhances the library system's robustness and functionality, addressing key aspects of book damage management and member accountability. By introducing mechanisms for reporting, repairing, and penalizing instances of book damage, the system promotes the responsible use of library resources and mitigates potential losses due to mishandling.

Analysis:

The added use cases contribute to the overall effectiveness and reliability of the library system by streamlining processes for handling damaged items and resolving outstanding fines. The integration of state machines enables systematic tracking of book conditions and member interactions, facilitating accurate status updates and informed decision-making.

Conclusion:

In conclusion, the extension and testing of the USE model for the library system demonstrates a proactive approach to software engineering, addressing real-world scenarios and ensuring system integrity. By incorporating additional functionalities and rigorously testing the implementation, the enhanced model offers an improved user experience and contributes to the overall efficiency of library operations.