SOFTWARE ENGINEERING 2

EXTENDING AND TESTING A COMPREHENSIVE USE MODEL FOR THE LIBRARY SYSTEM IN USE  
ASSIGNMENT REPORT

Jason Gaynor,   
Student ID C23409212  
  
Collaboration note: This model was created in collaboration with David Byrne C23308943

# 1. OVERVIEW

For this assignment, I selected the option to extend and test a comprehensive USE model for a Library System using the knowledge from lab sessions. This allowed me to use new real-world case scenarios such as book reservation, borrowing limits, fine payment, and more.  
  
The Library System code is a program that helps to keep track of books, copies, and members in a library system.   
Through this project, I expanded the starter code and Created the following:  
- Custom enums for tracking borrow/reserve status.  
- Refined class structure with new operations and constraints.  
- Implemented state machines for Book lifecycle.  
- Diagrams (Class, Object, State, Sequence).  
- Used SOIL scripts to test multiple scenarios.  
- Applied !openter / !opexit for method behavior validation.

# 2. IMPROVEMENTS MADE TO LIBRARY SYSTEM

Below are the main enhancements made to the initial lab model:

## New Enums

- BorrowStatus: Borrowed, NotBorrowed  
- ReserveStatus: Reserved, NotReserved  
> These Enums ensure the state of individual Copy instances.

## New Operations

- createCopy() in Book: Automates copy creation and links them via OfType.  
- borrow() / return() in both Book and Copy: Adjust availability counters and update state.  
- reserve() / removeReservation() in Copy: Handles copy reservation logic.  
- borrow() / return() in Person: Tracks member borrowing and limits.  
- viewBorrowed(): Lists all titles a member currently has.  
- payFine() in Person: Reduces fine balance after payment.  
- applyFine() in Employee: Increases a user’s fine (with constraint).

## State Machines

- Book: States include newTitle, available, unavailable, transitioned by create, borrow, or return.  
- Copy: Tracks reservation and loan status for transitions.

## Class and Associations

- Classes: Book, Copy, Person, Member, Employee  
- Associations:  
 - OfType between Book and Copy  
 - HasBorrowed between Person and Copy  
 - HasReserved between Copy and Person

## Constraints

I Implemented OCL constraints to enforce borrowing/reserving logic, reservation uniqueness, fine limits, and book/copy availability.

## SOIL Implementation

A script was created to incorporate Members, Employees, and Books; perform borrowing, reservation, and fine operations; and include invalid and edge case testing.

## Testing Enhancements

I Used !openter and !opexit to track method entry/exit and state mutation during payFine.

# 3. KEY FEATURES ADDED OR CHANGED

|  |  |
| --- | --- |
| Feature | Description |
| Reservation Logic | Now supports reserving specific copies and canceling them. |
| Fine Handling | Members (Jay,Dave) can pay fines; employees can apply them (limit: 50). (Tom being the employee in my example) |
| Borrowing Limits | Enforced constraints based on borrowed amount and copy availability. |
| Class Expansion | Separated Member and Employee from Person with role-specific behaviors. |
| SOIL Scenarios | Simulates all operations including edge cases like overpaying a fine. |
| State Machines  Use code model Library  enum BorrowStatus { Borrowed, NotBorrowed }  enum ReserveStatus { Reserved, NotReserved }  class Book  attributes  title : String  author : String  amount : Integer init = 2  available : Integer init = 2  operations  createCopy()  begin  declare c : Copy;  for i in Sequence{1..self.amount} do  self.available := self.amount;  c := new Copy;  c.borrowed := #NotBorrowed;  c.book := self;  c.reserved := #NotReserved;  insert(self, c) into OfType;  end  end  borrow()  begin  self.available := self.available - 1;  end  return()  begin  self.available := self.available + 1;  end  statemachines  psm States  states  newTitle : initial  available [available > 0]  unavailable [available = 0]  transitions  newTitle -> available { create }  available -> unavailable { [available = 1] borrow() }  available -> available { [available > 1] borrow() }  available -> available { return() }  unavailable -> available { return() }  end  end  class Copy  attributes  book : Book  borrowed : BorrowStatus init = #NotBorrowed  reserved : ReserveStatus init = #NotReserved  onLoan : Boolean  operations  borrow(p : Person)  begin  for p1 in self.reservation do  if p = p1 then  self.reserved := #NotReserved;  delete(self, p) from HasReserved;  end  end;  if self.reserved = #NotReserved then  insert(p, self) into HasBorrowed;  self.borrowed := #Borrowed;  self.book.borrow();  p.amountBorrowed := p.amountBorrowed + 1;  end  end  return(p : Person)  begin  delete(p, self) from HasBorrowed;  self.borrowed := #NotBorrowed;  self.book.return();  p.amountBorrowed := p.amountBorrowed - 1;  end  reserve(p : Person)  begin  self.reserved := #Reserved;  insert(self, p) into HasReserved;  WriteLine('This copy has been reserved for you');  end  removeReservation(p : Person)  begin  if self.reserved = #NotReserved then  WriteLine('This Copy does not have a reservation to remove');  else  self.reserved := #NotReserved;  delete(self, p) from HasReserved;  end  end  end  class Person  attributes  name : String  address : String  amountBorrowed : Integer init = 0  no\_onloan : Integer init = 0  limit : Integer init = 6  fine : Integer init = 0  status : String  operations  borrow(c : Copy)  begin  declare ok : Boolean;  ok := self.okToBorrow();  c.borrow(self);  end  okToBorrow() : Boolean  begin  if self.no\_onloan < 2 then  result := true  else  result := false  end  end  return(c : Copy)  begin  delete(self, c) from HasBorrowed;  self.no\_onloan := self.no\_onloan - 1;  c.return(self);  end  viewBorrowed()  begin  for c in self.borrowed do  WriteLine(c.book.title);  end;  end  payFine(amount : Integer)  reserve(c : Copy)  begin  c.reserve(self);  end  removeReservation(c : Copy)  begin  c.removeReservation(self);  end  end  class Employee < Person  attributes  employeeID : Integer  role : String  operations  applyFine(p : Person, amount : Integer)  begin  if p.fine + amount <= 50 then  p.fine := p.fine + amount;  else  WriteLine('Fine amount exceeds limit of 50');  end  end  end  class Member < Person  attributes  memberID : Integer  end  association OfType between  Book[1] role book  Copy[0..\*] role type  end  association HasBorrowed between  Person[0..1] role borrower  Copy[0..\*] role borrowed  end  association HasReserved between  Copy[0..1] role copy  Person[0..\*] role reservation  end  constraints  context Person::borrow(c : Copy)  pre underBorrowLimit : self.amountBorrowed < self.limit  pre copyNotYetBorrowed : self.borrowed -> excludes(c)  pre notDuplicateBook : self.borrowed.book -> excludes(c.book)  pre loanCapNotExceeded : self.no\_onloan < 2  context Copy::borrow(p : Person)  pre copyIsAvailable : self.borrowed = #NotBorrowed  context Book::borrow()  post availableNotNegative : self.available >= 0  context Person::return(c : Copy)  pre copyIsBorrowedByPerson : self.borrowed -> includes(c)  post copyIsReturned : self.borrowed -> excludes(c)  context Person::payFine(amount : Integer)  pre existingFine : self.fine > 0  post fineIsNonNegative : self.fine >= 0  context Person::reserve(c : Copy)  pre copyHasNoReservations : c.reservation -> isEmpty()  context Copy::reserve(p : Person)  pre copyNotReserved : self.reserved = #NotReserved  pre copyNotBorrowed : self.borrowed = #NotBorrowed  context Person::removeReservation(c : Copy)  pre reservationExists : c.reservation -> includes(self)  post reservationRemoved : c.reservation -> isEmpty()  context Employee::applyFine(p : Person, amount : Integer)  pre withinFineLimit : p.fine < 50  post stillWithinFineLimit : p.fine < 50   Soil Code  -- SOIL  !new Member('Dave')  !Dave.name := 'David Byrne'  !Dave.address := '7, O' Connell Street, Dublin'  !Dave.amountBorrowed := 3  !Dave.no\_onloan := 0  !Dave.limit := 6  !Dave.fine := 0  !Dave.status := 'Borrowed'  !Dave.memberID := 1234567  !new Member('Jay')  !Jay.name := 'Jason Gaynor'  !Jay.address := 'The Shop 133 Galtymore Rd, Drimnagh'  !Jay.amountBorrowed := 1  !Jay.no\_onloan := 1  !Jay.limit := 6  !Jay.fine := 0  !Jay.status := 'Borrowed'  !Jay.memberID := 014557324  !new Employee('Tom')  !Tom.name := 'Tommy Mustafa'  !Tom.address := 'The Academy Index, Dublin 1'  !Tom.amountBorrowed := 0  !Tom.no\_onloan := 0  !Tom.limit := 12  !Tom.fine := 0  !Tom.status := 'Reserved'  !Tom.employeeID := 123456789  !Tom.role := 'Librarian'  !new Book('PridePrejudice')  !PridePrejudice.title := 'Pride and Prejudice'  !PridePrejudice.author := 'Jane Austen'  !PridePrejudice.amount := 2  !PridePrejudice.available := 0  !PridePrejudice.createCopy()  !new Book('Dune')  !Dune.title := 'Dune'  !Dune.author := 'Frank Herbert'  !Dune.amount := 2  !Dune.available := 1  !Dune.createCopy()  !new Book('Sapiens')  !Sapiens.title := 'Sapiens: A Brief History of Humankind'  !Sapiens.author := 'Yuval Noah Harari'  !Sapiens.amount := 2  !Sapiens.available := 1  !Sapiens.createCopy()  !Dave.borrow(Copy1)  !Dave.borrow(Copy5)  !Jay.reserve(Copy4)  !Jay.removeReservation(Copy4)  !Jay.borrow(Copy4)  !Tom.reserve(Copy2)  !Copy4.onLoan := true  !Tom.applyFine(Jay, 30)  !openter Jay payFine(40)  !Jay.fine := (Jay.fine - 40)  !opexit  !Tom.applyFine(Jay, 100)  !Dave.borrow(Copy3) | Improves state visibility and system correctness for book availability. |

# 4. DIAGRAMS

- Class Diagram: Relationships between classes, enums, and associations.

A screenshot of a computer

AI-generated content may be incorrect.  
  
  
  
  
  
  
- Object Diagram: Snapshot of system state after running a scenario.A screenshot of a computer

AI-generated content may be incorrect.  
  
**Sequence diagrams**  
  
Sequence diagram for Tom employee reserving and applying fee  
A screenshot of a computer

AI-generated content may be incorrect.  
  
Sequence diagram, Dave borrowingA screenshot of a computer screen

AI-generated content may be incorrect.A screenshot of a calendar

AI-generated content may be incorrect.  
Sequence diagram for jay reserving, removing reservation ,borrowing and paying a fine  
A screenshot of a computer

AI-generated content may be incorrect.

- State Machine Diagrams: For Book lifecycle management.  
A screenshot of a computer

AI-generated content may be incorrect.

Use code for state machine  
A white background with black text

AI-generated content may be incorrect.  
  
  
  
  
  
  
  
  
  
**Constraints**  
A screenshot of a computer program

AI-generated content may be incorrect.

# 5. CONSTRAINTS TESTING (HIGHLIGHTS)

|  |  |
| --- | --- |
|  |  |
| TC1 - Book copy already reserved 1 |  |
|  |  |
|  |  |
|  |  |
| TC2 Copy of book already borrowed (4) |  |
| TC3 Cant return a book if it wasn’t borrowed before 3 |  |

# TC4 - Can’t apply fine over limit 6 A black screen with white text AI-generated content may be incorrect.

# TC5 – no reservation to be removed A computer screen shot of a black screen AI-generated content may be incorrect.

TC6 – cant over pay fine  
A computer screen with white text

AI-generated content may be incorrect.

# 6. CONCLUSION

The extended Library System model showcases an implementation of a real-world scenario using the USE tool. It integrates borrowing logic, fine management, and reservation workflows with enforcement through OCL constraints and state machines. This model supports consistent, testable behavior and ensures reliability in a multi-user environment. The project has deepened my understanding of modeling systems with precision and correctness.