William Paterson University
Department of Computer Science
CS 3500-001 Software Engineering Spring 2026

FINAL PROJECT DOCUMENTATION

Library Information and Administration System (LIAS)

05/11/2026

Team 04
Semih Aksehirli
Nihat Karahan
Mohammed Mostofa

Table of Contents

Project Introduction

Project Overview and System Goals

This document serves as the final, comprehensive deliverable for the Library Information and Administration System (LIAS), developed by Team 04 for a private liberal arts college library in the northeastern United States. This project encompasses the full software development lifecycle to provide the library with a robust, reliable, and fully automated system to manage its resources, patrons, and administrative operations.

What We Are Building

We are building a centralized library management system that serves as the primary platform for all library operations — from patron-facing catalog browsing and resource checkout, to administrator-level reporting and collection management. LIAS is designed to act as the interface between library users, the physical library collection, and the institution's external data systems.

The system's primary objectives are to:

- **Automate Core Library Operations:** Enable patrons to search the catalog, check out resources, place holds, and renew items online without manual staff intervention, reducing processing time and minimizing errors.
- **Enforce Library Business Rules:** Implement the library's specific policies, including patron-type checkout periods, overdue fine calculations at $0.25 per day, reserve restrictions, diploma holds for students with unpaid fines, and in-library-use-only enforcement for reserved materials.
- **Integrate with Institutional Systems:** Connect seamlessly to the college's Employee Information System (EIS) and Student Information System (SIS) for user authentication and account management, and to a nearby public library system for interlibrary loan fulfillment.
- **Empower Administrators:** Provide LIAS administrators with full control over the library catalog, patron accounts, magazine and journal subscriptions, and a suite of administrative reports covering collection statistics, resource usage, and subscription expirations.
- **Demonstrate Architectural Integrity:** The system is built using **Object-Oriented Design (OOD)** principles, ensuring a scalable, maintainable, and well-documented software structure.

Team Information

Team Members

This project was developed by Team 04: Semih Aksehirli, Nihat Karahan, and Mohammed Mostofa.

Experiences, Challenges and Lessons Learned

Semih Aksehirli	The biggest challenge was keeping all design decisions consistent as the project grew — what seemed like a small change in a class diagram would ripple into the communication diagrams and the implementation. I became much more careful about naming conventions and operation signatures as a result. Overall, the team stayed on track and each member carried their weight throughout the semester.
Nihat Karahan	The most valuable part of this project was seeing how the different phases of the software development life cycle connect in practice. Moving from use cases to class diagrams to communication diagrams made it clear that every earlier decision has consequences later on. Communication within the team was consistent, and when we ran into disagreements about design choices, we worked through them together and came out with better solutions.
Mohammed Mostofa	This project taught me how much effort goes into documenting a system properly before a single line of code is written. The Communication Diagrams were the hardest part for me personally — getting the sequence numbers right and making sure every object interaction was accounted for took a lot of careful work. By the end, the team had developed a solid rhythm and I am proud of what we produced together.

REQUIREMENTS & SPECIFICATION

A private liberal arts college library in the northeastern United States needs a new Library Information and Administration System (LIAS) to track and manage its resources. They wanted a computerized system to replace the current manual processes. The most obvious resource the library must manage is its books. Books are checked out, checked in, and requested by library patrons. Books can also have special status if they are placed on reserve or if they are reference books. In either case, such books may not leave the premises. Reminders are mailed to patrons when resources are more than two weeks overdue. Patrons are fined $0.25 per day that books are overdue. Students cannot obtain their diplomas if they have overdue charges that have not been cleared from the library.

The library also has other resources that may be checked out, including music CDs, software, audio tapes, and videos (VHS tapes and DVDs/BDs), each of which may only be checked out for one week at a time.

There are two types of patrons in this college library: faculty/staff and students. A student who works for a library is not classified as staff. Faculty/staff may check out books for three months, and students may check them out for 2 weeks. Any checkable library resource may be renewed as long as no other patron has requested it. Notice will be sent out when the resource is available to the patrons who put a hold on the resource. A requested resource will be held for two weeks for a patron. If it has not been checked out within two weeks, the resource will be returned to general circulation.

Faculty may place a book on reserve for the period of one semester, or they may bring in foreign resources (books, papers, disks, music CDs, magazines, video tapes, DVDs/BDs, or audio tapes that do not belong to the library) and put them on reserve.

The library must also manage a large selection of weekly, monthly, and quarterly magazines and journals, which may not be checked out but are available as reference materials. These magazines and journals are annually bound into volumes as reference books. Additional activities of the library staff include re-shelving books, renewing magazine/journal subscriptions, and ordering new library resources.

In automating the manual processes, LIAS should allow patrons to browse all the library resources online (users should be able to search the online catalog by author, subject, keyword, resource title, or Dewey call number). Patrons should be able to renew checkout items online. LIAS must also connect to the holdings of a nearby local public library so that interlibrary loan requests can be fulfilled.

Several library staff members are classified as LIAS administrators. The LIAS administrator maintains user information. His/her responsibility includes the acquisition and retirement of books in the library collection. In acquiring new books, a balance between meeting the requests of patrons and achieving a representative breadth in the collection is sought. Books are retired when their content is deemed to be out of date and of no historical value. Ideally, when a book is out of date, it will not be retired until a more up-to-date resource has replaced it in the library's collection.

LIAS should authenticate and distinguish the functionalities for two groups of users: library patrons (faculty/staff and students) and LIAS administrators.

LIAS will be able to generate some useful reports. Only the administrators can generate and access these reports. Examples of useful reports include expiration dates of magazine/journal subscriptions, number of books in each category (e.g., history, computer science, and so on), book usage (so that additional copies of the same book may be needed), library statistics (patrons, newspapers, magazines, journals, books, or any other library resources), etc.

There are two external database systems that LIAS must connect to. One is called EIS (Employee Information System) that consists of all the faculty and staff information. The other system is called SIS (Student Information System), which has all the students' information.

ACTORS

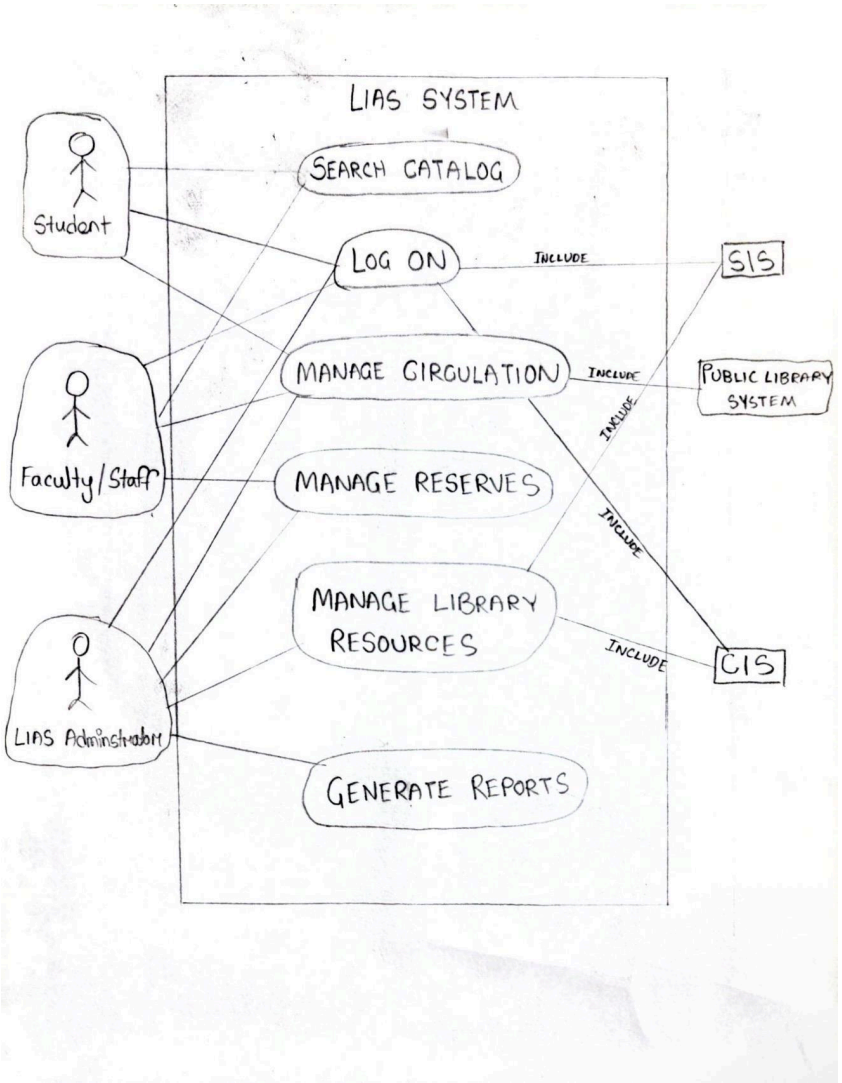
The following actors interact with LIAS:

Actor	Description
Student	Students enrolled at the college. Students may check out books for two weeks. Students with uncleared overdue charges cannot obtain their diplomas.
Faculty/Staff	Faculty and staff members of the college. Faculty may check out books for three months. Faculty may also place books or foreign resources on reserve for one semester. A student who works at the library is not classified as staff.
LIAS Administrator	Library staff members designated as LIAS administrators. They are responsible for maintaining user information, acquiring and retiring books, managing library resources, renewing magazine/journal subscriptions, and generating system reports.
SIS (External System)	Student Information System – An external database system that contains all student information. LIAS connects to SIS to verify student records and manage student patron accounts.

EIS (External System)	Employee Information System – An external database system that contains all faculty and staff information. LIAS connects to EIS to verify and retrieve patron information for faculty and staff accounts.
Public Library System (External)	A nearby local public library whose holdings LIAS connects to in order to fulfill interlibrary loan requests from patrons.

USE CASE DIAGRAM

The Use Case Diagram below illustrates the relationships between all actors and the 6 use cases identified for LIAS. Patron use cases cover day-to-day library operations accessible to students and faculty/staff. Administrative use cases are restricted to Library Staff and LIAS Administrators. External systems (SIS, EIS, and the Public Library System) participate as secondary actors through system-to-system integrations.



USE CASE DESCRIPTIONS

The following section provides detailed Use Case Descriptions for all use cases identified in the LIAS Use Case Diagram. Each description includes the actor(s) involved, pre-conditions, main dialog flow, post-conditions, and any dependency relationships to other use cases. Operations are shown in **`bold red Courier New font`** using the format **`[ClassName]:operationName(parameters)`**.

- **Name: LOGON**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - User has navigated to the LIAS portal.
  - User has a valid User ID and Password.

- **Dialog:**
  - System displays the logon screen and requests the User ID and Password.
    `[Screen]:display_logon_screen()`
    `[Screen]:capture_userID_PW()`
  - System queries EIS (for Faculty/Staff) or SIS (for Students) via <<include>> to authenticate the user and determine the appropriate role.

    `[DB_Interface]:communicate_DB()`
    `[DB_Interface]:authenticate(userID, PW, OUT status):user_type`
  - If authentication is successful, the system displays the correct dashboard based on the user's role: Student, Faculty/Staff, or LIAS Administrator.

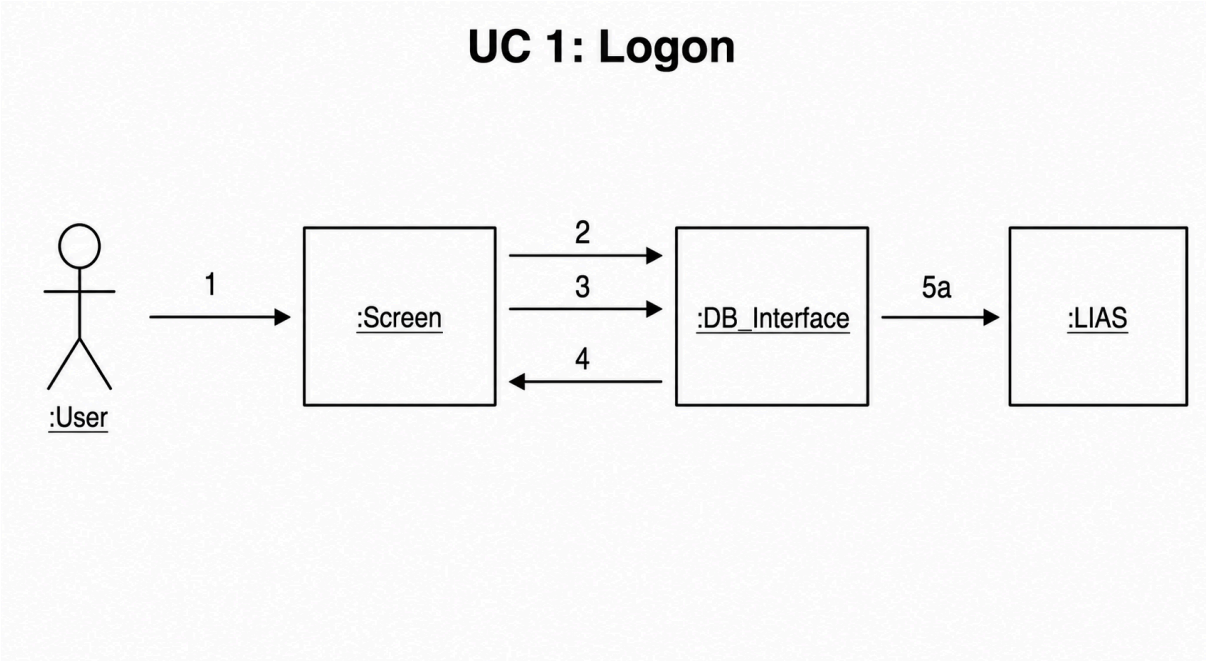
    `[LIAS]:grant_access(user_type)`
    `[Screen]:display_dashboard(user_type)`
  - If authentication fails, the system allows up to 5 retry attempts.

    `[LIAS]:increment_retry_count()`
    `[Screen]:display_error_message()`
  - After 5 unsuccessful attempts, the session is locked and a help message is displayed.

    `[LIAS]:lock_session()`
    `[Screen]:display_help_message()`

- **Post-Conditions:**
  - If authentication is successful, the user is logged in and granted role-appropriate access.
  - If authentication fails after 5 attempts, the session is locked and a help or error message is displayed.

Communication Diagram




```
1: display_logon_screen() / capture_userID_PW()

2: [internal] Screen displays dashboard or error message

3: communicate_DB()

4: authenticate(userID, PW, OUT status) : user_type

5a: [success] grant_access(user_type) ->
display_dashboard(user_type)

5b: [failure] display_error_message() / [5 attempts lock
session()] + display_help_message()

6: display_dashboard(user_type) [returned from LIAS to Screen]
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Successful Login	1, 3, 4, 5a, 6, 2	User enters valid credentials; authenticated via EIS/SIS; dashboard displayed.
Failed Login (< 5 tries)	1, 3, 4, 5b, 2	Invalid credentials; error shown; retry count incremented; user may try again.
Account Locked (5 tries)	1, 3, 4, 5b (x5), 2	After 5 failures, session locked; help message displayed via Error_Handling.

- **Name: SEARCH CATALOG**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - User is successfully logged into LIAS.
- **Dialog:**
  - The system displays the online catalog search page.

    ```
    [Screen]:display_catalog_screen()
    ```

  - The user can search for resources by author, subject, keyword, title, or Dewey call number.

    ```
    [Screen]:capture_search_criteria(searchType, searchValue)
    ```

  - System queries the catalog database to find matching resources.

    ```
    [DB_Interface]:search_catalog(searchType, searchValue):resultList
    [DB_Interface]:communicate_DB()
    ```

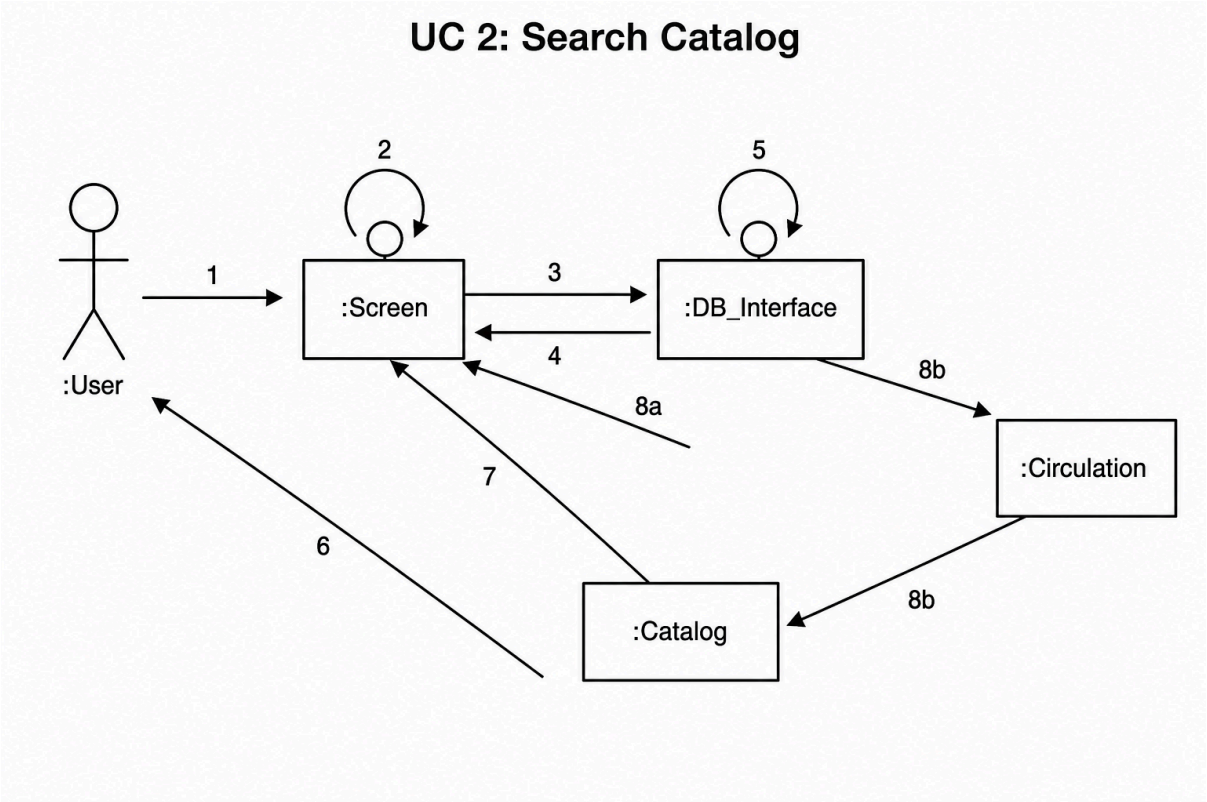
  - The system displays matching materials along with their current status (Available, Checked Out, On Reserve, Reference Only).

    ```
    [Screen]:display_search_results(resultList)
    [Catalog]:get_resource_status(resourceID):status
    ```

  - Patron may place a hold or proceed to check out from the search results, connecting to MANAGE CIRCULATION.

    ```
    [Circulation]:place_hold(patronID, resourceID)
    [Screen]:display_resource_details(resourceID)
    ```

- **Post-Conditions:**
  - The user can review the search results and may continue to MANAGE CIRCULATION to borrow or place a hold on an item.




```
1: display_catalog_screen()
2: capture_search_criteria(searchType, searchValue)
3: communicate_DB()
4: search_catalog(searchType, searchValue) : resultList
5: display_search_results(resultList)
6: get_resource_status(resourceID) : status
7: display_resource_details(resourceID) [Screen internal, shown in loop]
8a: [hold requested] place_hold(patronID, resourceID)
8b: display_hold_confirmation()
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Search – Results Found	1, 2, 3, 4, 5, 6, 7	Query returns matches; resources shown with status.
Search – No Results	1, 2, 3, 4, 5	No matches; empty result or informational message displayed.
Place Hold from Results	1, 2, 3, 4, 5, 6, 7, 8a, 8b	Patron selects checked-out resource and places hold; confirmation shown.

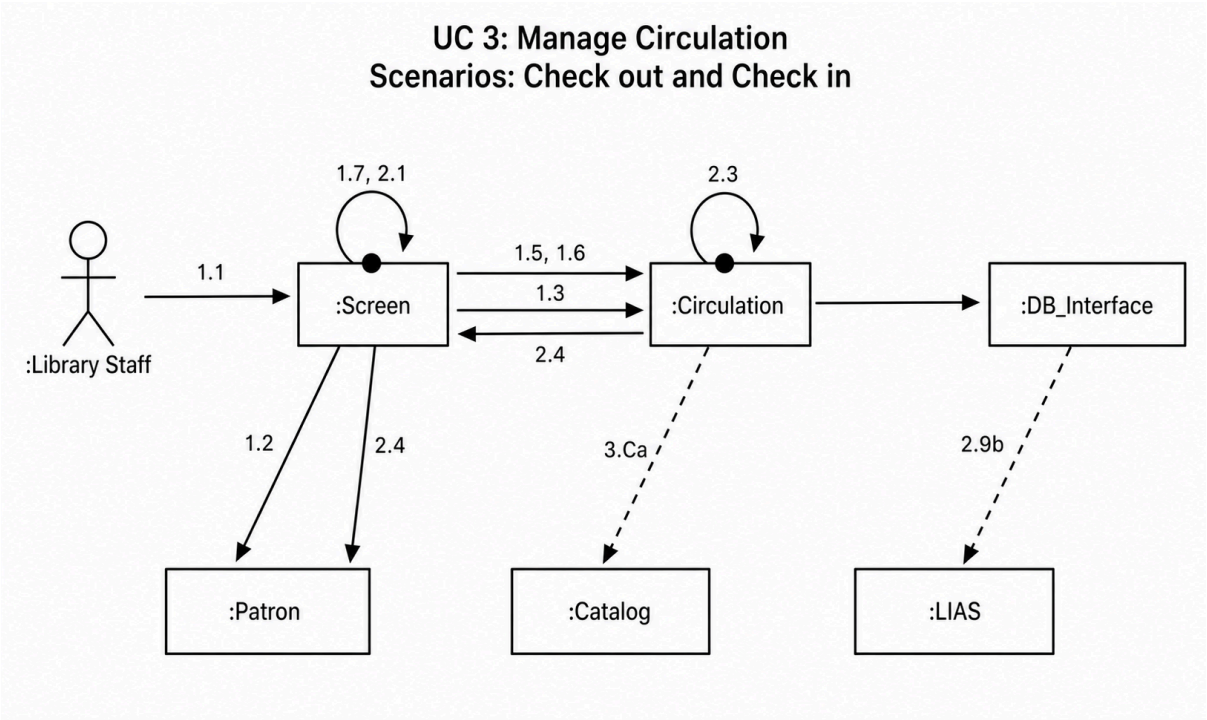
- **Name: MANAGE CIRCULATION**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - User (Patron) has successfully logged on (LOGON Use Case) OR Library Staff is operating the circulation desk terminal.
  - The resource to be checked out, checked in, renewed, or held is present in the LIAS catalog.
  - LIAS has connectivity to the central library database.
- **Dialog:**
  - CHECK OUT:
    - Library Staff scans or enters the resource's barcode/call number and the patron's library ID.
      `[Screen]:capture_checkout_info(resourceBarcode, patronID)`
    - System validates patron status (no outstanding blocks: unpaid fines, diploma hold, etc.).
      `[Patron]:validate_patron_status(patronID):blockStatus [DB_Interface]:communicate_DB()`
    - System assigns checkout period based on patron type and resource type: Faculty/Staff: Books check out for 3 months. CDs, Software, Audio Tapes, and Videos check out for 1 week. Students: Books check out for 2 weeks. CDs, Software, Audio Tapes, and Videos check out for 1 week.
      `[Circulation]:assign_checkout_period(patronType, resourceType):dueDate`
    - Reference and Reserve materials cannot be checked out.
      `[Catalog]:check_resource_restriction(resourceID):restrictionType`
    - System records the checkout transaction and displays the due date to the patron.
      `[Circulation]:record_checkout(patronID, resourceID, dueDate)`
      `[Screen]:display_due_date(dueDate)`
  - CHECK IN:
    - Library Staff scans or enters the returned resource's barcode.
      `[Screen]:capture_checkin_barcode(resourceBarcode)`
    - System records the return date and calculates any overdue fines ($0.25 per day late).
      `[Circulation]:record_checkin(resourceID, returnDate)`
      `[Circulation]:calculate_fine(dueDate, returnDate):fineAmount`

- If the resource has a pending hold, system flags it as 'On Hold' and notifies the waiting patron via email; hold is reserved for 2 weeks.
  `[Circulation]:check_hold_queue(resourceID):nextPatronID`
  `[LIAS]:send_hold_notification_email(patronID, resourceID)`
- If no holds exist, resource status is updated to 'Available' and item is queued for re-shelving.
  `[Catalog]:update_resource_status(resourceID, status)`
  `[Circulation]:queue_for_reshelving(resourceID)`

- RENEW:
  - Patron submits a renewal request online (via SEARCH CATALOG) or at the circulation desk.
    `[Screen]:capture_renewal_request(patronID, resourceID)`
  - System checks whether any other patron has placed a hold on the resource.
    `[Circulation]:check_hold_exists(resourceID):boolean`
  - If no holds exist: renewal is granted; due date is extended by the standard checkout period.
    `[Circulation]:grant_renewal(patronID, resourceID):newDueDate`
    `[Screen]:display_renewal_confirmation(newDueDate)`
  - If holds exist: renewal is denied; patron is notified and instructed to return the resource by the original due date.
    `[Circulation]:deny_renewal(patronID, resourceID)`
    `[LIAS]:send_renewal_denial_email(patronID)`
- PLACE HOLD / REQUEST:
  - Patron places a hold on a checked-out or unavailable resource (via SEARCH CATALOG).
    `[Circulation]:place_hold(patronID, resourceID)`
    `[DB_Interface]:communicate_DB()`
  - System adds patron to the hold queue and confirms the request.
    `[Circulation]:add_to_hold_queue(patronID, resourceID)`
    `[Screen]:display_hold_confirmation()`

        - When the resource is returned, system notifies the next patron in the hold queue via email.
          `[LIAS]:send_hold_notification_email(patronID, resourceID)`
        - If the patron does not check out the resource within 2 weeks of notification, the hold is cancelled and the resource returns to general circulation.
          `[Circulation]:expire_hold(patronID, resourceID)`
          `[Catalog]:update_resource_status(resourceID, "Available")`
    - OVERDUE REMINDERS & FINES:
        - The system automatically generates and mails a reminder when a resource is more than 2 weeks overdue.
          `[Circulation]:check_overdue_resources()`
          `[LIAS]:send_overdue_reminder_email(patronID, resourceID)`
        - A fine of $0.25 per day is assessed for each overdue resource.
          `[Circulation]:calculate_fine(dueDate, currentDate):fineAmount`
          `[Patron]:apply_fine(patronID, fineAmount)`
        - Outstanding fines are flagged on the patron's account; student accounts with unpaid fines are reported to the Registrar, blocking diploma issuance.
          `[Patron]:flag_unpaid_fines(patronID)`
          `[LIAS]:report_diploma_hold_to_SIS(studentID)`
    - RE-SHELVING:
        - Upon check-in, Library Staff marks the returned resource as 'Pending Re-shelving' in LIAS.
          `[Catalog]:update_resource_status(resourceID, "Pending Re-shelving")`
        - The system maintains a re-shelving queue listing all items awaiting return to their designated shelf location (organized by Dewey call number).
          `[Circulation]:get_reshelving_queue():queueList`
          `[Screen]:display_reshelving_queue(queueList)`
        - Library Staff confirms re-shelving completion in LIAS.
          `[Circulation]:confirm_reshelving(resourceID)`
          `[Catalog]:update_resource_status(resourceID, "Available")`
- **Post-Conditions:**
    - Checkout transaction is recorded with patron ID, resource ID, and due date.
    - Check-in transaction is recorded; fines are calculated and applied to the patron's account if applicable.

- Returned resources are added to the re-shelving queue; status updated to 'Available' once Library Staff confirms re-shelving.
- Renewal is recorded with a new due date, or denial is logged and the patron is notified.
- Hold request is queued; patron is notified when the resource becomes available.
- Overdue notices and fine records are updated in the patron's account in the central database.



1.1: capture_checkout_info(resourceBarcode, patronID)

1.2: validate_patron_status(patronID) : blockStatus

1.3: communicate_DB()

1.4: check_resource_restriction(resourceID) : restrictionType

1.5: assign_checkout_period(patronType, resourceType) : dueDate

1.6: record_checkout(patronID, resourceID, dueDate)

1.7: display_due_date(dueDate)

2.1: capture_checkin_barcode(resourceBarcode)

2.2: record_checkin(resourceID, returnDate)

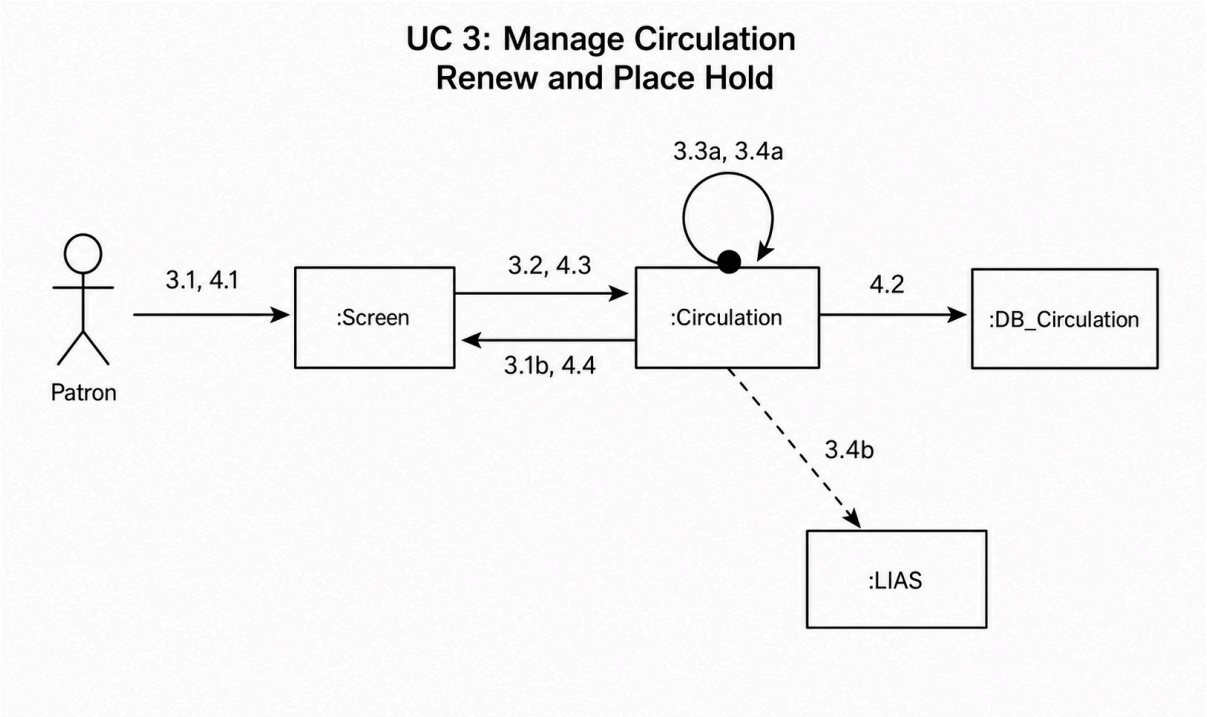
2.3: calculate_fine(dueDate, returnDate) : fineAmount

2.4: apply_fine(patronID, fineAmount)

2.5b: send_hold_notification_email(patronID, resourceID)

2.6a: update_resource_status(resourceID, 'Available')

queue_for_reshelving(resourceID)



3.1: capture_renewal_request(patronID, resourceID)

3.2: check_hold_exists(resourceID) : boolean

3.3a: [no hold] grant_renewal(patronID, resourceID) : newDueDate

3.3b: display_renewal_confirmation(newDueDate)

3.4a: [hold exists] deny_renewal(patronID, resourceID)

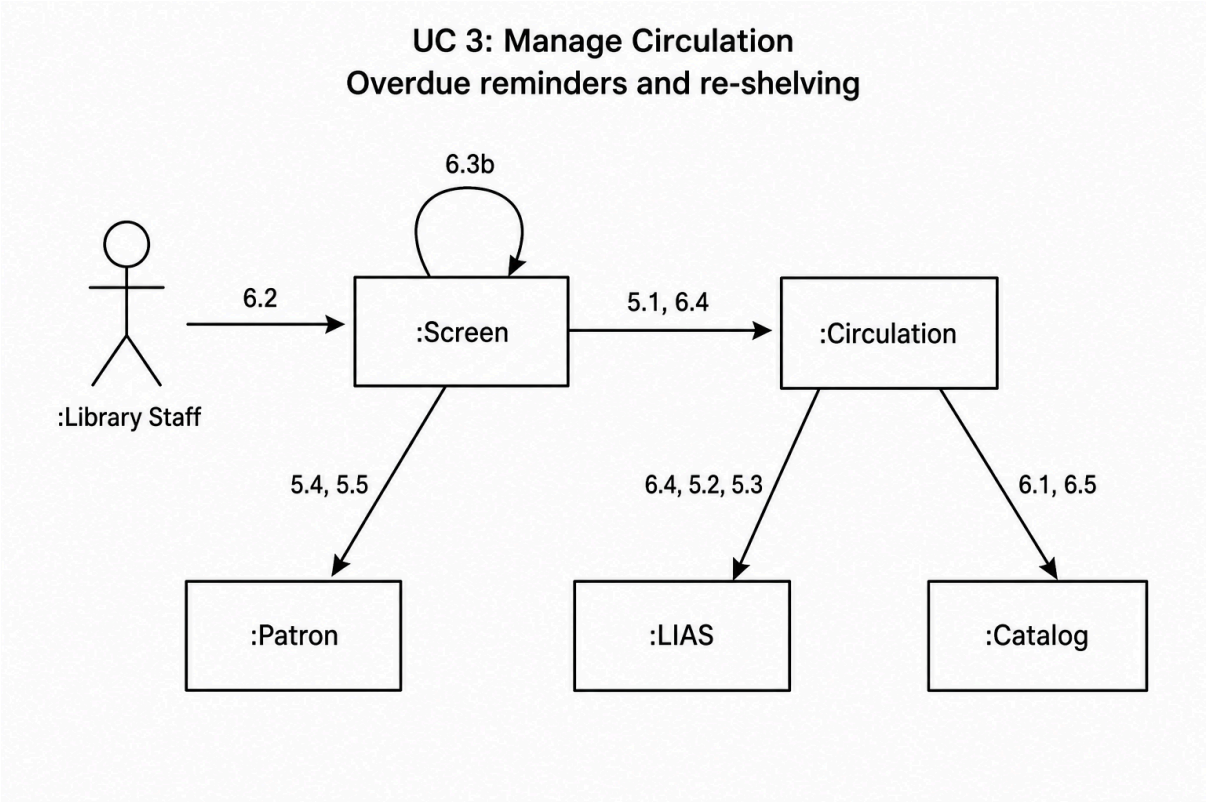
3.4b: send_renewal_denial_email(patronID)

4.1: place_hold(patronID, resourceID)

4.2: communicate_DB()

4.3: add_to_hold_queue(patronID, resourceID)

4.4: display_hold_confirmation()




```
5.1: check_overdue_resources()
5.2: send_overdue_reminder_email(patronID, resourceID)
5.3: calculate_fine(dueDate, currentDate) : fineAmount
5.4: apply_fine(patronID, fineAmount)
5.5: flag_unpaid_fines(patronID)
5.6: [student] report_diploma_hold_to_SIS(studentID)
6.1: update_resource_status(resourceID, 'Pending Re-shelving')
6.2: get_reshelving_queue() : queueList
6.3: display_reshelving_queue(queueList)
6.4: confirm_reshelving(resourceID)
6.5: update_resource_status(resourceID, 'Available')
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Check Out – Success	1.1→1.7	No blocks; no restriction; checkout recorded; due date displayed.
Check Out – Patron Blocked	1.1, 1.2	Patron has unpaid fines or diploma hold; checkout denied.
Check Out – Restricted	1.1, 1.2, 1.3, 1.4	Resource is Reference or Reserve; checkout denied.

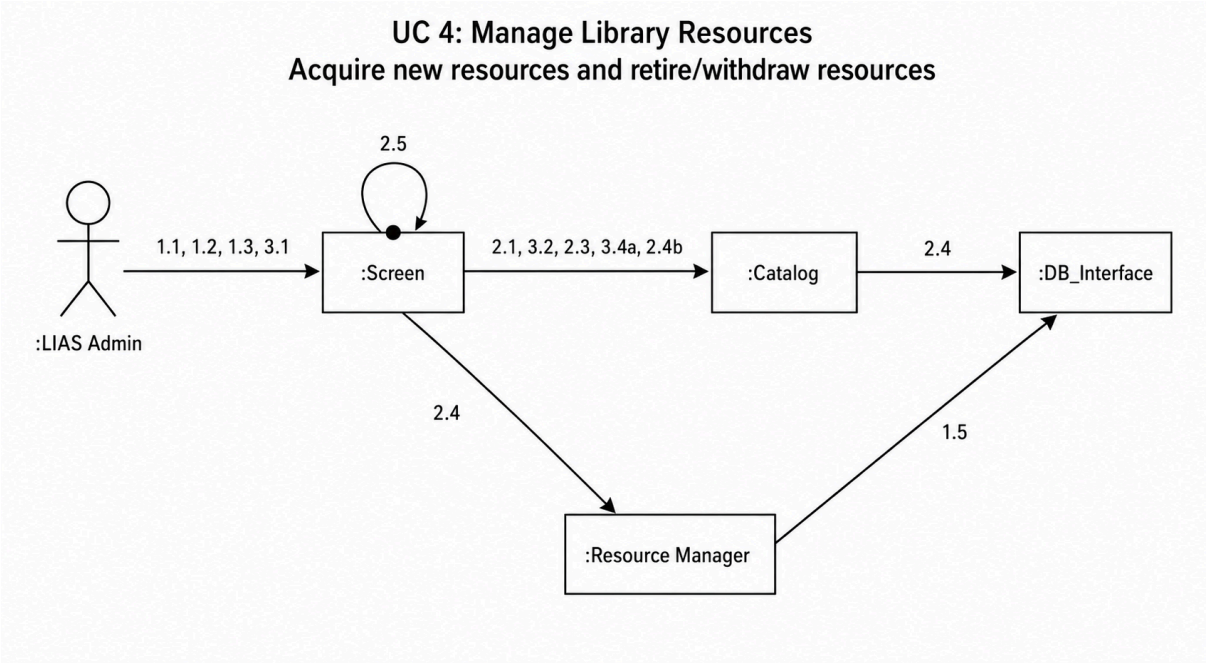
Check In – No Hold	**2.1→2.4, 2.6a, 2.6b**	Return recorded; fine computed; status set Available; queued for re-shelving.
Check In – With Hold	**2.1→2.4, 2.5a, 2.5b**	Hold patron notified by email; status set On Hold.
Renew – Granted	**3.1, 3.2, 3.3a, 3.3b**	No holds; renewal granted; new due date displayed.
Renew – Denied	**3.1, 3.2, 3.4a, 3.4b**	Hold exists; denial logged; patron emailed original due date.
Place Hold	**4.1→4.4**	Hold added to queue; confirmation shown to patron.
Overdue Reminders & Fines	**5.1→5.6**	Overdue check runs; reminders sent; fines applied; diploma hold for students.
Re-Shelving	**6.1→6.5**	Item queued; staff retrieves queue; confirms; status set Available.

- **Name: MANAGE LIBRARY RESOURCES**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - LIAS Administrator has successfully logged on (LOGON Use Case) with Administrator privileges.
  - LIAS is connected to the central library database.
  - For ordering, an external vendor or supplier database connection is available.
- **Dialog:**
  - ACQUIRE NEW RESOURCES:
    - Administrator reviews patron requests and evaluates collection breadth gaps.
      `[Screen]:display_acquisition_dashboard()`
      `[Catalog]:get_patron_requests():requestList`
    - Administrator searches supplier catalogs and selects titles or resources to acquire.
      `[Screen]:capture_acquisition_selection(resourceInfo)`
    - The system generates and tracks purchase orders for new books, CDs, magazines, software, audio tapes, and videos.
      `[ResourceManager]:generate_purchase_order(resourceInfo):orderID`
      `[DB_Interface]:communicate_DB()`
    - Upon receipt, the Administrator adds the new resource record to the LIAS catalog (title, author, call number, ISBN/barcode, format, quantity, location).
      `[Catalog]:add_resource(title, author, callNumber, ISBN, format, quantity, location)`
      `[Catalog]:update_resource_status(resourceID, "Available")`
  - UPDATE / EDIT RESOURCE RECORDS:

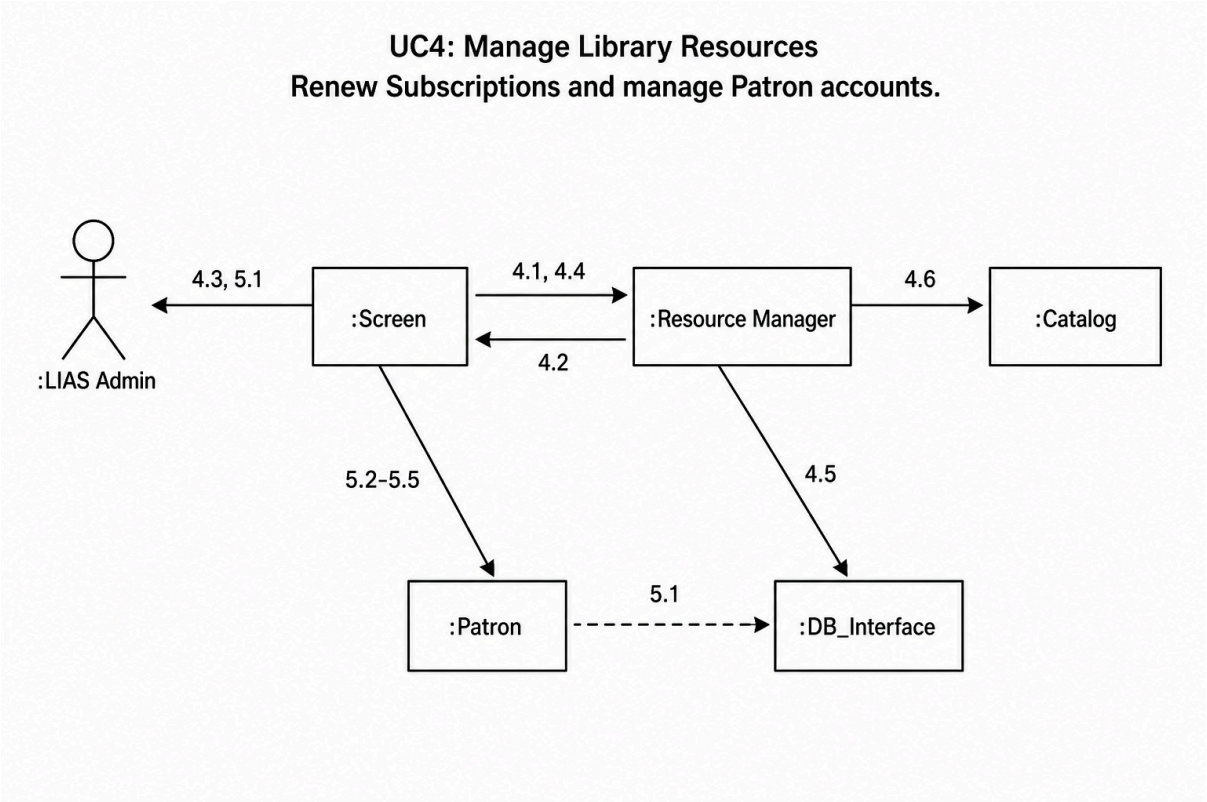
- Administrator searches for an existing resource by title, call number, or barcode.
  `[DB_Interface]:search_catalog(searchType, searchValue):resultList`
- Administrator edits metadata: title, author, price, edition, location, quantity, or format.
  `[Screen]:capture_resource_edits(resourceID, updatedFields)`
- System saves the updated record to the central database.
  `[Catalog]:update_resource_record(resourceID, updatedFields)  [DB_Interface]:communicate_DB()`

- RETIRE / WITHDRAW RESOURCES:
  - Administrator identifies resources that are out of date or of no historical value.
    `[Screen]:display_retirement_candidates():candidateList`
  - System verifies the resource is not currently checked out and that a replacement resource exists in the collection.
    `[Catalog]:verify_retirement_eligibility(resourceID):eligible`
    `[Catalog]:check_replacement_exists(resourceID):boolean`
  - If conditions are met, the Administrator confirms retirement; the system marks the record as 'Withdrawn' and removes it from public search results.
    `[Catalog]:retire_resource(resourceID)`
    `[Catalog]:update_resource_status(resourceID, "Withdrawn")`
  - If no replacement exists, the system warns the Administrator prior to retirement.
    `[Screen]:display_retirement_warning(resourceID)`
- RENEW MAGAZINE / JOURNAL SUBSCRIPTIONS:
  - System displays a list of periodicals with upcoming subscription expiration dates.
    `[ResourceManager]:get_expiring_subscriptions():subList`
    `[Screen]:display_subscription_list(subList)`
  - Administrator reviews the list and selects subscriptions to renew or cancel.
    `[Screen]:capture_subscription_decision(subID, decision)`

        - System records renewal/cancellation and updates the subscription register.
          `[ResourceManager]:update_subscription(subID, decision)  [DB_Interface]:communicate_DB()`
        - Annually, expired volume issues are bound into reference volumes and added to the catalog.
          `[Catalog]:add_bound_volume(volumeInfo)`
    - MANAGE PATRON ACCOUNTS:
        - Administrator adds new patron accounts by importing or manually entering data from EIS (Faculty/Staff) or SIS (Students).
          `[DB_Interface]:communicate_DB()`
          `[Patron]:create_patron_account(patronInfo, patronType)`
        - Administrator deactivates or removes accounts for patrons who are no longer affiliated with the college.
          `[Patron]:deactivate_patron_account(patronID)`
        - Administrator resets patron passwords upon request and manages fine/fee records.
          `[Patron]:reset_patron_password(patronID)`
          `[Patron]:manage_fine_records(patronID)`
- **Post-Conditions:**
    - New resource records are added to the LIAS catalog and are searchable by patrons.
    - Updated resource records reflect current metadata in the catalog.
    - Retired resources are removed from public catalog search and flagged as 'Withdrawn' in the database.
    - Subscription renewals are recorded and expiration dates are updated.
    - Patron account changes are saved and synchronized with EIS/SIS as appropriate.




```
1.1: display_acquisition_dashboard()

1.2: get_patron_requests() : requestList

1.3: capture_acquisition_selection(resourceInfo)

1.4: generate_purchase_order(resourceInfo) : orderID

1.5: communicate_DB()

1.6: add_resource(title, author, callNumber, ISBN, format,
qty, location)

1.7: update_resource_status(resourceID, 'Available')

2.1-2.3: search_catalog() / capture_resource_edits() /
update_resource_record() / communicate_DB()

3.1: display_retirement_candidates()

3.2: verify_retirement_eligibility(resourceID) : boolean

3.3: check_replacement_exists(resourceID) : boolean

3.4a: retire_resource()

3.4b: update_resource_status('Withdrawn')
3.5: display_retirement_warning
```




```
4.1: get_expiring_subscriptions() : subList
4.2: display_subscription_list(subList)
4.3: capture_subscription_decision(subID, decision)
4.4: update_subscription(subID, decision)
4.5: communicate_DB()
4.6: [annually] add_bound_volume(volumeInfo)
5.1: communicate_DB() [query EIS/SIS for patron data]
5.2: create_patron_account(patronInfo, patronType)
5.3: deactivate_patron_account(patronID)
5.4: reset_patron_password(patronID)
5.5: manage_fine_records(patronID)
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Acquire New Resource	1.1→1.7	Admin reviews requests, generates PO, adds received item to catalog.
Edit Resource Record	2.1→2.4	Admin searches record, edits metadata, system saves to DB.

Retire Resource (eligible)	**3.1→3.4b**	Not checked out, replacement exists; marked Withdrawn.
Retire (no replacement)	**3.1, 3.2, 3.3, 3.5**	No replacement; system warns admin before proceeding.
Renew Subscription	**4.1→4.6**	Expiring subscriptions listed; admin decides; system records.
Manage Patron Account	**5.1→5.5**	Admin creates, deactivates, resets, or manages fine records synced with EIS/SIS.

- **Name: MANAGE RESERVES**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - Faculty/Staff or LIAS Administrator has successfully logged on (LOGON Use Case) with the appropriate role privileges.
  - The resource to be placed on reserve exists in the LIAS catalog, or the faculty member has a foreign resource ready to register.
  - LIAS has connectivity to the central library database.
- **Dialog:**
  - *PLACE LIBRARY RESOURCE ON RESERVE:*
    - Faculty/Staff searches for and selects a library-owned book or resource via the SEARCH CATALOG use case.

      `[DB_Interface]:search_catalog(searchType, searchValue):resultList`

    - Faculty/Staff submits a reserve request and specifies the reserve duration (one semester).

      `[Screen]:capture_reserve_request(facultyID, resourceID, duration)`

    - System verifies the patron's Faculty/Staff role (students may not place items on reserve).

      `[Patron]:verify_faculty_role(patronID):boolean`

    - System updates the resource status to "On Reserve" and records the faculty member's ID, the resource ID, and the reservation end date.

      `[ReserveManager]:create_reserve_record(facultyID, resourceID,endDate)`
      `[Catalog]:update_resource_status(resourceID, "On Reserve")`

    - The reserved item may not leave the premises for the duration of the reserve period.

      `[Catalog]:set_inlibrary_restriction(resourceID)`

- *REGISTER A FOREIGN RESOURCE ON RESERVE:*
  - Faculty/Staff presents a foreign resource (personal books, papers, disks, music CDs, magazines, video tapes, DVDs/BDs, or audio tapes not owned by the library) to Library Staff.

    `[Screen]:capture_foreign_resource_info(resourceInfo, facultyID)`
  - LIAS Administrator or Library Staff creates a temporary catalog record for the foreign resource, noting that it is not library-owned.

    `[Catalog]:add_foreign_resource(resourceInfo, isForeignOwned:true)`
  - System sets the resource status to "On Reserve – Foreign" and assigns the same one-semester duration and in-library-use-only restriction.

    `[ReserveManager]:create_reserve_record(facultyID, resourceID, endDate)`
    `[Catalog]:update_resource_status(resourceID, "On Reserve -Foreign")`
- *RESERVE EXPIRATION:*
  - System automatically monitors reserve end dates.

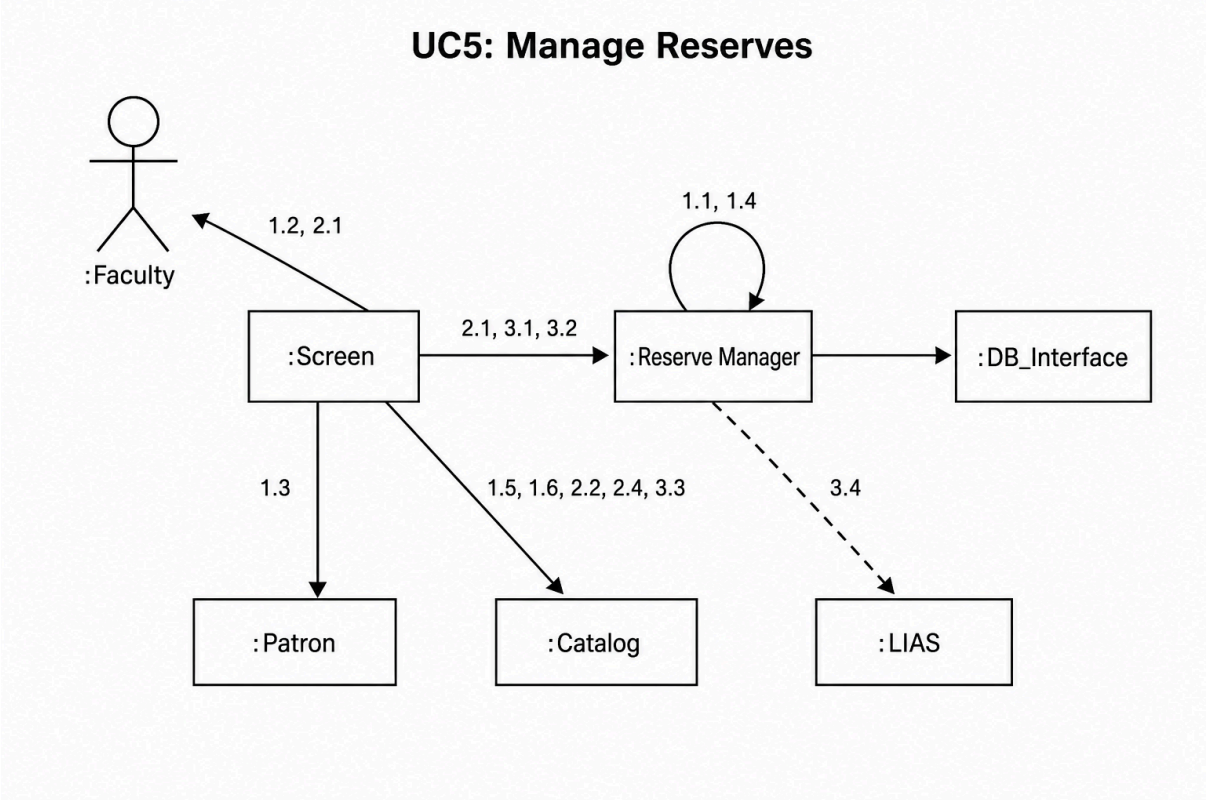
    `[ReserveManager]:check_reserve_expirations()`
  - Upon expiration, system updates the resource status: library-owned items return to general circulation (or "Available"), and foreign resources are flagged for return to the faculty member.

    `[ReserveManager]:expire_reserve(resourceID, reserveType)`
    `[Catalog]:update_resource_status(resourceID, status)`
  - System notifies the faculty member that the reserve period has ended.

    `[LIAS]:send_reserve_expiration_email(facultyID, resourceID)`

- **Post-Conditions:**
  - The reserved resource is restricted to in-library use only and cannot be checked out for the duration of the reserve period.
  - The reserve record is stored in the LIAS database with the faculty member's ID, resource ID, reserve type (library-owned or foreign), and expiration date.

- Upon expiration, library-owned resources are returned to general circulation and foreign resources are flagged for retrieval by the faculty member.




```
1.1: search_catalog(searchType, searchValue) : resultList
1.2: capture_reserve_request(facultyID, resourceID, duration)
1.3: verify_faculty_role(patronID) : boolean
1.4: create_reserve_record(facultyID, resourceID, endDate)
1.5: update_resource_status(resourceID, 'On Reserve')
1.6: set_inlibrary_restriction(resourceID)
2.1: capture_foreign_resource_info(resourceInfo, facultyID)
2.2: add_foreign_resource(resourceInfo, isForeignOwned: true)
2.3: create_reserve_record()
2.4: update_resource_status(resourceID, 'On Reserve - Foreign')
3.1: check_reserve_expirations()
3.2: expire_reserve(resourceID, reserveType)
3.3a: [library] update_status('Available')
3.3b: [foreign] update_status('Flagged for Return')
3.4: send_reserve_expiration_email
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Place Library Resource on Reserve	1.1→1.6	Faculty searches, submits reserve; role verified; record created; status On Reserve.
Role Verification Fails	1.1, 1.2, 1.3	Non-faculty patron; verify_faculty_role returns false; reserve denied.
Register Foreign Resource	2.1→2.4	Faculty brings personal material; temp catalog entry created; On Reserve - Foreign.
Expiration – Library-Owned	3.1, 3.2, 3.3a, 3.4	Expired; status set Available; faculty emailed.
Expiration – Foreign Resource	3.1, 3.2, 3.3b, 3.4	Expired; status flagged for retrieval; faculty emailed.

- **Name: GENERATE REPORTS**
- **Author: Semih Aksehirli, Nihat Karahan, Mohammed Mostofa**
- **Last Update: 4/13/2026**
- **Pre-Conditions:**
  - LIAS Administrator has successfully logged on (LOGON Use Case) with Administrator privileges.
  - LIAS has connectivity to the central library database.
  - Relevant data (patron records, resource inventory, subscription records, circulation history) exists in the LIAS database.
- **Dialog:**
  - Administrator navigates to the Reports module from the Administrator dashboard.

    `[Screen]:display_reports_dashboard()`
    `[ReportGenerator]:get_available_reports():reportTypeList`
  - System displays available report types: Subscription Expiration, Collection by Category, Book Usage,
    Library Statistics.
    `[Screen]:display_report_menu(reportTypeList)`
  - Administrator selects the desired report type and optionally applies filters (e.g., date range, category, patron type).

    `[Screen]:capture_report_selection(reportType, filters)`
  - System queries the LIAS database and compiles the results into a structured report.

    `[ReportGenerator]:generate_subscription_expiration_report(filters):reportData`

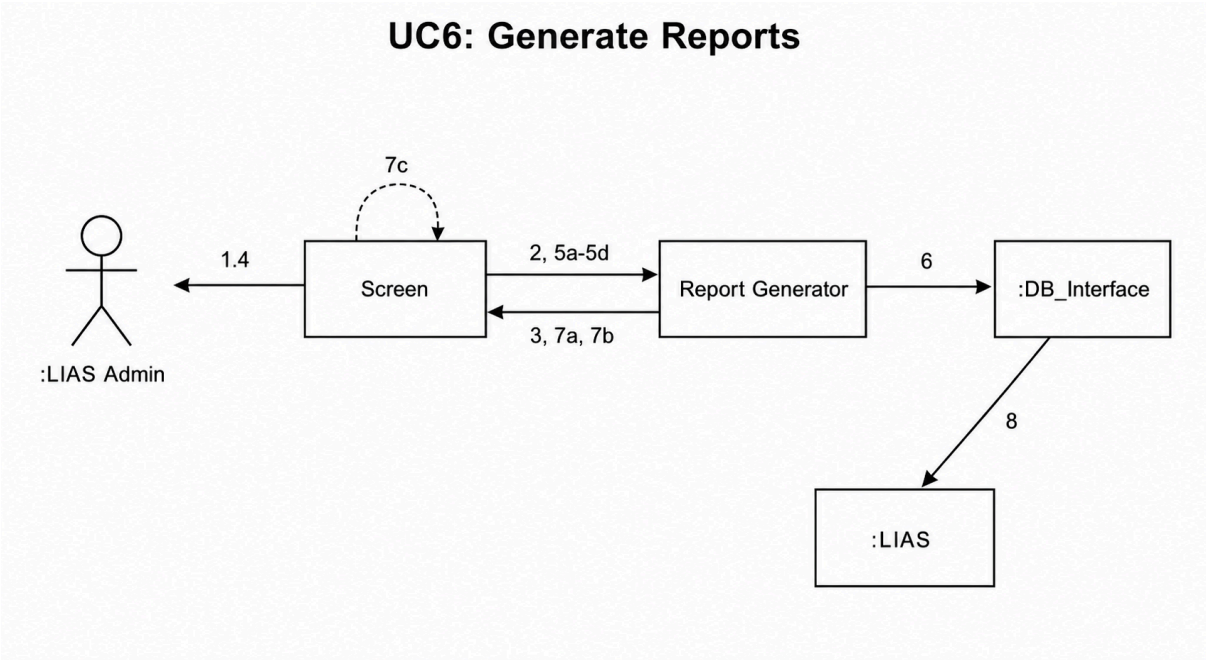
`[ReportGenerator]:generate_collection_category_report(filters):reportData`
`[ReportGenerator]:generate_book_usage_report(filters):reportData`
`[ReportGenerator]:generate_library_statistics_report(filters):reportData`
`[DB_Interface]:communicate_DB()`

- Administrator reviews the report on-screen or exports it (e.g., PDF or print format) for administrative use.

  [Screen]:display_report(reportData)
  [ReportGenerator]:export_report(reportData, format)
- If insufficient data exists to generate the selected report, the system displays an informational message.

  [Screen]:display_info_message(message)
  [LIAS]:log_report_access(adminID, reportType)

- **Post-Conditions:**
  - The requested report is generated and displayed or exported for the Administrator's review.
  - No data in the LIAS database is modified as a result of report generation.
  - Report access is logged in the system audit trail, restricted to LIAS Administrators only.



`1: display_reports_dashboard()`

`2: get_available_reports() : reportTypeList`

```
3: display_report_menu(reportTypeList)

4: capture_report_selection(reportType, filters)

5a: generate_subscription_expiration_report(filters) : reportData

5b: generate_collection_category_report(filters) : reportData

5c: generate_book_usage_report(filters) : reportData

5d: generate_library_statistics_report(filters) : reportData

6: communicate_DB()

7a: display_report(reportData)

7b: export_report(reportData, format)

7c: [no data] display_info_message(message)

8: log_report_access(adminID, reportType)
```

Scenario Sequence

Scenario	Sequence Numbers	Description
Subscription Expiration Report	1, 2, 3, 4, 5a, 6, 7a, 8	Admin requests subscription report; DB queried; expiring subs listed.
Collection by Category Report	1, 2, 3, 4, 5b, 6, 7a, 8	Dewey category filter applied; resource counts per category displayed.
Book Usage Report	1, 2, 3, 4, 5c, 6, 7a, 8	Circulation frequencies compiled; high-demand titles identified.
Library Statistics Report	1, 2, 3, 4, 5d, 6, 7a, 8	Summary count of all patrons and resources generated and displayed.
Export Report	1, 2, 3, 4, 5*, 6, 7b, 8	Report compiled then exported as PDF or print format.
Insufficient Data	1, 2, 3, 4, 5*, 6, 7c	No data available; informational message shown; access still logged.

CLASS DIAGRAMS AND IMPLEMENTATION

Class Diagram

The following classes were identified by analyzing the operations derived from all six Use Case Descriptions above. Each class is presented in UML notation with its name, attributes (where applicable), and operations. A description paragraph follows each class diagram.

Screen
Handles all user interface rendering and input capture for LIAS. The Screen class is responsible for displaying every view the user interacts with — from the logon screen and catalog search page to dashboards, error messages, and reports. It also captures all user input and forwards it to the appropriate system classes.
Attributes
- x_coordinate : int = 0
- y_coordinate : int = 0
- color : hex = FFFFFF
- border : int = 1
- pixel : int
- brightness : int
- sharpness : int
Operations
+ display_logon_screen()
- capture_userID_PW()
+ display_dashboard(user_type : string)
+ display_error_message()
+ display_help_message()
+ display_catalog_screen()
+ capture_search_criteria(searchType : string, searchValue : string)
+ display_search_results(resultList : list)
+ display_resource_details(resourceID : integer)
+ capture_checkout_info(resourceBarcode : string, patronID : integer)
+ display_due_date(dueDate : date)
+ capture_checkin_barcode(resourceBarcode : string)
+ capture_renewal_request(patronID : integer, resourceID : integer)
+ display_renewal_confirmation(newDueDate : date)

+ display_hold_confirmation()
+ display_reshelving_queue(queueList : list)
+ display_acquisition_dashboard()
+ capture_acquisition_selection(resourceInfo : string)
+ capture_resource_edits(resourceID : integer, updatedFields : list)
+ display_retirement_candidates() : candidateList
+ display_retirement_warning(resourceID : integer)
+ display_subscription_list(subList : list)
+ capture_subscription_decision(subID : integer, decision : string)
+ capture_reserve_request(facultyID : integer, resourceID : integer, duration : string)
+ capture_foreign_resource_info(resourceInfo : string, facultyID : integer)
+ display_reports_dashboard()
+ display_report_menu(reportTypeList : list)
+ capture_report_selection(reportType : string, filters : list)
+ display_report(reportData : object)
+ display_info_message(message : string)
+ get/set_coordinate()
+ get/set_color()
+ get/set_resolution()
+ get/set_brightness()
+ get/set_sharpness()

Class implementation

```
class Screen {

  public:

    void display_logon_screen();                                    //
Displays the LIAS logon screen to the user

    void display_dashboard(string user_type);                       //
Displays role-appropriate dashboard after successful login

    void display_error_message();                                   //
Displays an error or system message to the user

    void display_help_message();                                    //
Displays help message after 5 failed login attempts

    void display_catalog_screen();                                  //
Displays the catalog search screen
```

```
    void capture_search_criteria(string searchType, string searchValue); //
Captures search type and keyword from user

    void display_search_results(list resultList);                        //
Displays list of resources matching the search

    void display_resource_details(int resourceID);                       //
Displays full details of a single resource

    void capture_checkout_info(string resourceBarcode, int patronID); //
Captures barcode and patron ID to begin checkout

    void display_due_date(date dueDate);                                 //
Displays the assigned due date after checkout

    void capture_checkin_barcode(string resourceBarcode);                //
Captures barcode of the item being returned

    void capture_renewal_request(int patronID, int resourceID);          //
Captures patron and resource ID for a renewal request

    void display_renewal_confirmation(date newDueDate);                  //
Displays new due date after a successful renewal

    void display_hold_confirmation();                                     //
Displays confirmation that a hold has been placed

    void display_reshelving_queue(list queueList);                       //
Displays the current re-shelving queue to staff

    void display_acquisition_dashboard();                                //
Displays the resource acquisition management screen

    void capture_acquisition_selection(string resourceInfo);             //
Captures admin's selection of resource to acquire

    void capture_resource_edits(int resourceID, list updatedFields);  //
Captures admin's edits to resource metadata

    list display_retirement_candidates();                                //
Displays list of resources eligible for retirement

    void display_retirement_warning(int resourceID);                     //
Warns admin when no replacement edition exists

    void display_subscription_list(list subList);                        //
Displays expiring subscriptions to admin

    void capture_subscription_decision(int subID, string decision);   //
Captures admin's renew or cancel decision

    void capture_reserve_request(int facultyID, int resourceID, string
duration); // Captures faculty reserve request details

    void capture_foreign_resource_info(string resourceInfo, int facultyID);
// Captures details of a faculty-owned item

    void display_reports_dashboard();                                     //
Displays the reports management screen

    void display_report_menu(list reportTypeList);                       //
Displays available report types to admin
```

```
    void capture_report_selection(string reportType, list filters);   //
Captures admin's report type and filter selection

    void display_report(object reportData);                            //
Displays the generated report on screen

   void display_info_message(string message);                         //
Displays informational message when no data is available

    void get_coordinate();                                              //
Returns the current x and y screen coordinates

    void set_coordinate();                                              //
Sets the x and y screen coordinates

    void get_color();                                                   //
Returns the current screen color value

    void set_color();                                                   //
Sets the screen color value

    void get_resolution();                                              //
Returns the current screen resolution setting

    void set_resolution();                                              //
Sets the screen resolution

    void get_brightness();                                              //
Returns the current brightness level

    void set_brightness();                                              //
Sets the brightness level

    void get_sharpness();                                               //
Returns the current sharpness level

    void set_sharpness();                                               //
Sets the sharpness level


  private:

    void capture_userID_PW();  // Internally captures user ID and password
input from the logon screen

    int x_coordinate;          // X-axis position of the screen window,
default = 0

    int y_coordinate;          // Y-axis position of the screen window,
default = 0

    hex color;                 // Screen color value in hexadecimal, default
= FFFFFF

    int border;                // Border thickness of the screen window,
default = 1

    int pixel;                 // Pixel resolution value of the screen

    int brightness;            // Screen brightness level

    int sharpness;             // Screen sharpness level
```

```
};
```

DB_Interface
Manages all communication between LIAS and its central database, as well as connections to external systems (EIS, SIS, and the Public Library System). This class acts as the data access layer for the entire system, responsible for authenticating users, executing catalog queries, and persisting all transactions to the database.
Attributes
`+ status : string { validated, invalidated }`
Operations
`+ communicate_DB()`
`+ authenticate(userID : string, PW : string, OUT status : string) : user_type`
`+ search_catalog(searchType : string, searchValue : string) : resultList`

Class implementation

```
class DB_Interface {

  public:

    void communicate_DB();
// Establishes and maintains connection to the central database

    string authenticate(string userID, string PW, OUT string status);
// Validates user credentials; sets OUT status to "validated" or
"invalidated"; returns user_type

    list search_catalog(string searchType, string searchValue);
// Queries the database for resources matching the search type and value;
returns resultList


    string status;  // Current authentication status: "validated" or
"invalidated"

};
```

LIAS
The central system class for the Library Information and Administration System. LIAS coordinates high-level session management, role-based access control, and all email notification services. It also serves as the integration point with external systems (SIS) for diploma hold reporting and manages the audit trail for report access.
Operations
+ grant_access(user_type : string)
+ increment_retry_count()
+ lock_session()
+ send_hold_notification_email(patronID : integer, resourceID : integer)
+ send_renewal_denial_email(patronID : integer)
+ send_overdue_reminder_email(patronID : integer, resourceID : integer)
+ report_diploma_hold_to_SIS(studentID : integer)
+ send_reserve_expiration_email(facultyID : integer, resourceID : integer)
+ log_report_access(adminID : integer, reportType : string)
+ logout()

Class implementation

```
class LIAS {

  public:

    void grant_access(string user_type);
// Grants role-appropriate system access after successful authentication

    void increment_retry_count();
// Increments the failed login attempt counter by one

    void lock_session();
// Locks the user session after 5 consecutive failed login attempts

    void send_hold_notification_email(int patronID, int resourceID);
// Sends email to patron when their held resource becomes available

    void send_renewal_denial_email(int patronID);
// Sends denial notification email to patron when renewal is refused

    void send_overdue_reminder_email(int patronID, int resourceID);
// Sends overdue reminder email to the patron responsible for the item

    void report_diploma_hold_to_SIS(int studentID);
// Reports a student diploma hold status to the SIS external system

    void send_reserve_expiration_email(int facultyID, int resourceID);
// Notifies faculty by email when their reserve period has expired

    void log_report_access(int adminID, string reportType);
// Logs the report access event to the system audit trail
```

```
    void logout();
// Terminates the current user session
};
```

Patron
Represents all registered library users, encompassing both Faculty/Staff and Student sub-types. The Patron class manages account lifecycle (creation, deactivation, password reset), stores patron profile data, tracks fine balances, and enforces role-based restrictions such as diploma holds for students with unpaid fines.
Attributes
- patronID : integer {5 digits}
- firstName : string = blank
- lastName : string = blank
- email : string
- patronType : string { Student, Faculty, Staff }
- fineBalance : dollar = 0.00
- accountStatus : string { Active, Locked, Deactivated }
- diplomaHold : boolean = false
Operations
+ validate_patron_status(patronID : integer) : blockStatus
+ verify_faculty_role(patronID : integer) : boolean
+ apply_fine(patronID : integer, fineAmount : dollar)
+ flag_unpaid_fines(patronID : integer)
+ create_patron_account(patronInfo : object, patronType : string)
+ deactivate_patron_account(patronID : integer)
+ reset_patron_password(patronID : integer)
+ manage_fine_records(patronID : integer)
+ getPatronID() : integer
+ setPatronID(id : integer)
+ getPatronType() : string
+ setPatronType(type : string)
+ getFineBalance() : dollar
+ setFineBalance(amount : dollar)

Class implementation

```
class Patron {

  public:

    string validate_patron_status(int patronID);          // Returns
blockStatus indicating whether patron has blocks such as unpaid fines or
diploma hold

    bool verify_faculty_role(int patronID);               // Returns true if
the patron has faculty role privileges

    void apply_fine(int patronID, dollar fineAmount);     // Applies the
specified fine amount to the patron's account balance

    void flag_unpaid_fines(int patronID);                 // Flags the
patron's account for having an unpaid fine balance

    void create_patron_account(object patronInfo, string patronType); //
Creates a new patron account synced with EIS or SIS

    void deactivate_patron_account(int patronID);         // Deactivates an
existing patron account in the system

    void reset_patron_password(int patronID);             // Resets the login
password for the specified patron

    void manage_fine_records(int patronID);               // Views or adjusts
the fine records for a patron

    int getPatronID();                                    // Returns the
patron's unique five-digit ID

    void setPatronID(int id);                             // Sets the
patron's unique ID

    string getPatronType();                               // Returns the
patron type: Student, Faculty, or Staff

    void setPatronType(string type);                      // Sets the patron
type

    dollar getFineBalance();                              // Returns the
patron's current outstanding fine balance in dollars

    void setFineBalance(dollar amount);                   // Sets the
patron's fine balance


  private:

    int patronID;          // Unique five-digit identifier assigned to the
patron

    string firstName;      // Patron's first name, default = blank

    string lastName;       // Patron's last name, default = blank

    string email;          // Patron's email address used for system
notifications
```

```
    string patronType;      // Account type: Student, Faculty, or Staff
    dollar fineBalance;     // Current outstanding fine amount in dollars,
default = 0.00
    string accountStatus;   // Account standing: Active, Locked, or
Deactivated
    bool diplomaHold;       // True if a diploma hold has been placed on a
student account, default = false
};
```

Catalog
Manages the complete inventory of all library resources. The Catalog class is responsible for adding, updating, retiring, and tracking every item in the library's collection — books, CDs, software, audio tapes, videos, magazines, journals, and foreign reserve materials. It maintains the current status of each resource and enforces restrictions such as reference-only and in-library-use-only designations.
Attributes
- resourceID : integer
- title : string
- author : string
- callNumber : string
- ISBN : string
- format : string { Book, CD, Software, AudioTape, Video, Magazine, Journal }
- quantity : integer
- location : string
- status : string { Available, CheckedOut, OnReserve, OnHold, Reference, Withdrawn, PendingReshelving }
- isForeignOwned : boolean = false
Operations
+ get_resource_status(resourceID : integer) : status
+ update_resource_status(resourceID : integer, status : string)
+ check_resource_restriction(resourceID : integer) : restrictionType
+ add_resource(title : string, author : string, callNumber : string, ISBN : string, format : string, quantity : integer, location : string)
+ add_foreign_resource(resourceInfo : object, isForeignOwned : boolean)
+ update_resource_record(resourceID : integer, updatedFields : list)
+ retire_resource(resourceID : integer)
+ verify_retirement_eligibility(resourceID : integer) : boolean
+ check_replacement_exists(resourceID : integer) : boolean

+ set_inlibrary_restriction(resourceID : integer)
+ add_bound_volume(volumeInfo : object)
+ get_patron_requests() : requestList
+ getResourceID() : integer
+ setResourceID(id : integer)
+ getTitle() : string
+ setTitle(title : string)

Class implementation

```
class Catalog {

  public:

    string get_resource_status(int resourceID);
// Returns the current circulation status of the specified resource

    void update_resource_status(int resourceID, string status);
// Updates the circulation status of the specified resource in the database

    string check_resource_restriction(int resourceID);
// Returns the restriction type for the resource such as Reference or None

    void add_resource(string title, string author, string callNumber, string
ISBN,

                      string format, int quantity, string location);
// Adds a newly acquired resource record to the catalog

    void add_foreign_resource(object resourceInfo, bool isForeignOwned);
// Registers a faculty-owned item as a temporary catalog entry

    void update_resource_record(int resourceID, list updatedFields);
// Updates specified metadata fields for an existing resource record

    void retire_resource(int resourceID);
// Marks the resource as Withdrawn in the catalog

    bool verify_retirement_eligibility(int resourceID);
// Returns true if the resource is not currently checked out

    bool check_replacement_exists(int resourceID);
// Returns true if a newer edition of the resource exists in the catalog

    void set_inlibrary_restriction(int resourceID);
// Restricts the resource to in-library use only

    void add_bound_volume(object volumeInfo);
// Adds an annually bound journal volume to the catalog

    list get_patron_requests();
// Retrieves the list of pending patron resource acquisition requests
```

```
    int getResourceID();
// Returns the unique resource ID

    void setResourceID(int id);
// Sets the resource ID

    string getTitle();
// Returns the full title of the resource

    void setTitle(string title);
// Sets the title of the resource


  private:

    int resourceID;      // Unique identifier assigned to each resource at the
time of cataloging

    string title;        // Full title of the resource

    string author;       // Author or creator of the resource

    string callNumber;   // Library call number using LC or Dewey Decimal
classification

    string ISBN;         // International Standard Book Number of the resource

    string format;       // Media type: Book, CD, Software, AudioTape, Video,
Magazine, or Journal

    int quantity;        // Number of physical copies held by the library

    string location;     // Physical shelf location of the resource in the
library

    string status;       // Circulation status: Available, CheckedOut,
OnReserve, OnHold, Reference, Withdrawn, or PendingReshelving

    bool isForeignOwned; // True if the item is a faculty-owned resource on
foreign reserve, default = false

};
```

Handles all transactional operations related to the borrowing and return of library resources. The Circulation class manages checkouts, check-ins, renewals, holds, fine calculations, and the re-shelving queue. It acts as the core engine for day-to-day library operations, enforcing checkout periods and hold policies defined in the requirements.
Attributes
- transactionID : integer
- patronID : integer
- resourceID : integer
- checkoutDate : date
- dueDate : date
- returnDate : date
- fineAmount : dollar = 0.00
- renewalCount : integer = 0
Operations
+ assign_checkout_period(patronType : string, resourceType : string) : dueDate
+ record_checkout(patronID : integer, resourceID : integer, dueDate : date)
+ record_checkin(resourceID : integer, returnDate : date)
+ calculate_fine(dueDate : date, returnDate : date) : fineAmount
+ check_hold_queue(resourceID : integer) : nextPatronID
+ queue_for_reshelving(resourceID : integer)
+ check_hold_exists(resourceID : integer) : boolean
+ grant_renewal(patronID : integer, resourceID : integer) : newDueDate
+ deny_renewal(patronID : integer, resourceID : integer)
+ place_hold(patronID : integer, resourceID : integer)
+ add_to_hold_queue(patronID : integer, resourceID : integer)
+ expire_hold(patronID : integer, resourceID : integer)
+ check_overdue_resources()
+ get_reshelving_queue() : queueList
+ confirm_reshelving(resourceID : integer)
+ getTransactionID() : integer
+ getDueDate() : date
+ setDueDate(date : date)

Class implementation

```
class Circulation {
```

```
  public:
    date assign_checkout_period(string patronType, string resourceType); //
Calculates and returns the due date based on patron and resource type
    void record_checkout(int patronID, int resourceID, date dueDate);    //
Saves a new checkout transaction record to the database
    void record_checkin(int resourceID, date returnDate);                //
Records the check-in of a returned resource with its actual return date
    dollar calculate_fine(date dueDate, date returnDate);                //
Calculates the overdue fine at a rate of $0.25 per day; returns fineAmount
    int check_hold_queue(int resourceID);                                //
Returns the nextPatronID from the hold queue for the specified resource
    void queue_for_reshelving(int resourceID);                           //
Adds a returned item to the re-shelving queue
    bool check_hold_exists(int resourceID);                              //
Returns true if another patron has a pending hold on this resource
    date grant_renewal(int patronID, int resourceID);                    //
Approves the renewal and returns the new due date
    void deny_renewal(int patronID, int resourceID);                     //
Logs a denial when a hold exists on the resource
    void place_hold(int patronID, int resourceID);                       //
Creates a hold request for the patron on the specified resource
    void add_to_hold_queue(int patronID, int resourceID);                //
Adds the patron to the resource's hold waiting queue
    void expire_hold(int patronID, int resourceID);                      //
Expires an unfulfilled hold request
    void check_overdue_resources();                                       //
Scans the system for all currently overdue resources
    list get_reshelving_queue();                                         //
Retrieves the current list of items to be re-shelved; returns queueList
    void confirm_reshelving(int resourceID);                             //
Staff confirms that the resource has been returned to its shelf location
    int getTransactionID();                                              //
Returns the unique ID for this circulation transaction
    date getDueDate();                                                   //
Returns the assigned due date for this transaction
    void setDueDate(date date);                                          //
Sets the due date for this transaction


  private:
    int transactionID;   // Unique identifier for each circulation
transaction
```

```
    int patronID;          // ID of the patron involved in this transaction
    int resourceID;        // ID of the resource involved in this transaction
    date checkoutDate;     // Date the resource was checked out
    date dueDate;          // Date by which the resource must be returned
    date returnDate;       // Actual date the resource was returned
    dollar fineAmount;     // Fine amount charged for this transaction,
default = 0.00
    int renewalCount;      // Number of times this transaction has been
renewed, default = 0
};
```

ReserveManager
Manages the complete lifecycle of reserve records for both library-owned and foreign resources. The ReserveManager class creates and stores reserve records, monitors expiration dates, and coordinates with the Catalog class to update resource statuses upon reserve creation and expiration. It enforces the one-semester duration limit and in-library-use-only restriction for all reserved materials.
Attributes
- reserveID : integer
- facultyID : integer
- resourceID : integer
- reserveType : string { Library-Owned, Foreign }
- startDate : date
- endDate : date
- status : string { Active, Expired }
Operations
+ create_reserve_record(facultyID : integer, resourceID : integer, endDate : date)
+ check_reserve_expirations()
+ expire_reserve(resourceID : integer, reserveType : string)
+ getReserveID() : integer
+ getFacultyID() : integer
+ getEndDate() : date
+ setEndDate(date : date)

Class implementation:

```
class ReserveManager {
```

```
  public:
    void create_reserve_record(int facultyID, int resourceID, date endDate);
// Creates a new reserve record in the database with the specified expiration
date
    void check_reserve_expirations();
// Scans all reserve records and identifies those that have expired
    void expire_reserve(int resourceID, string reserveType);
// Processes the expiration based on whether the reserve is Library-Owned or
Foreign
    int getReserveID();
// Returns the unique ID for this reserve record
    int getFacultyID();
// Returns the faculty member's ID associated with this reserve
    date getEndDate();
// Returns the expiration date of the reserve period
    void setEndDate(date date);
// Sets the expiration date of the reserve period


  private:
    int reserveID;       // Unique identifier for this reserve record
    int facultyID;       // ID of the faculty member who submitted the
reserve request
    int resourceID;      // ID of the resource placed on reserve
    string reserveType;  // Type of reserve: Library-Owned or Foreign
    date startDate;      // Date the reserve period began
    date endDate;        // Date the reserve period expires
    string status;       // Current status of the reserve: Active or Expired
};
```

ResourceManager
Handles administrative resource management tasks including acquisition of new library resources and management of magazine and journal subscriptions. The ResourceManager class generates and tracks purchase orders, manages subscription renewal and cancellation decisions, and coordinates the annual binding of expired periodical volumes into reference works.
Attributes
- orderID : integer
- subscriptionID : integer
- publicationTitle : string
- subscriptionType : string { Weekly, Monthly, Quarterly }

- expirationDate : date
- renewalStatus : string { Active, Renewed, Cancelled }
Operations
+ generate_purchase_order(resourceInfo : object) : orderID
+ get_expiring_subscriptions() : subList
+ update_subscription(subID : integer, decision : string)
+ getOrderID() : integer
+ getSubscriptionID() : integer
+ getExpirationDate() : date
+ setExpirationDate(date : date)

Class implementation

```
class ResourceManager {

  public:

    int generate_purchase_order(object resourceInfo);        // Generates a
purchase order for the selected resource; returns orderID

    list get_expiring_subscriptions();                       // Retrieves the
list of subscriptions nearing their expiration date; returns subList

    void update_subscription(int subID, string decision);    // Saves the
admin's renew or cancel decision for the specified subscription

    int getOrderID();                                        // Returns the
unique ID of the purchase order

    int getSubscriptionID();                                 // Returns the
unique ID of the subscription record

    date getExpirationDate();                                // Returns the
expiration date of the subscription

    void setExpirationDate(date date);                       // Sets the
expiration date of the subscription


  private:

    int orderID;             // Unique identifier for each purchase order
generated

    int subscriptionID;      // Unique identifier for each subscription
record

    string publicationTitle; // Title of the periodical or journal
subscription

    string subscriptionType; // Frequency type: Weekly, Monthly, or Quarterly

    date expirationDate;     // Date the subscription expires
```

```
    string renewalStatus;    // Current status: Active, Renewed, or Cancelled
};
```

ReportGenerator
Responsible for querying the LIAS database and compiling all administrative reports accessible to LIAS Administrators. The ReportGenerator class supports four report types: Subscription Expiration, Collection by Category, Book Usage, and Library Statistics. It also provides export functionality and retrieves the list of available report types for display.
Operations
`+ get_available_reports() : reportTypeList`
`+ generate_subscription_expiration_report(filters : list) : reportData`
`+ generate_collection_category_report(filters : list) : reportData`
`+ generate_book_usage_report(filters : list) : reportData`
`+ generate_library_statistics_report(filters : list) : reportData`
`+ export_report(reportData : object, format : string)`

Class implementation

```
class ReportGenerator {

  public:

    list get_available_reports();
// Retrieves and returns the full list of available report types; returns
reportTypeList

    object generate_subscription_expiration_report(list filters);
// Queries the database and compiles subscription expiration data; returns
reportData

    object generate_collection_category_report(list filters);
// Queries the database and compiles collection data grouped by Dewey
category; returns reportData

    object generate_book_usage_report(list filters);
// Queries the database and compiles checkout frequency data for all
resources; returns reportData

    object generate_library_statistics_report(list filters);
// Queries the database and compiles overall patron and resource summary
statistics; returns reportData

    void export_report(object reportData, string format);
// Exports the compiled report as a PDF file or sends it to the system
printer

};
```