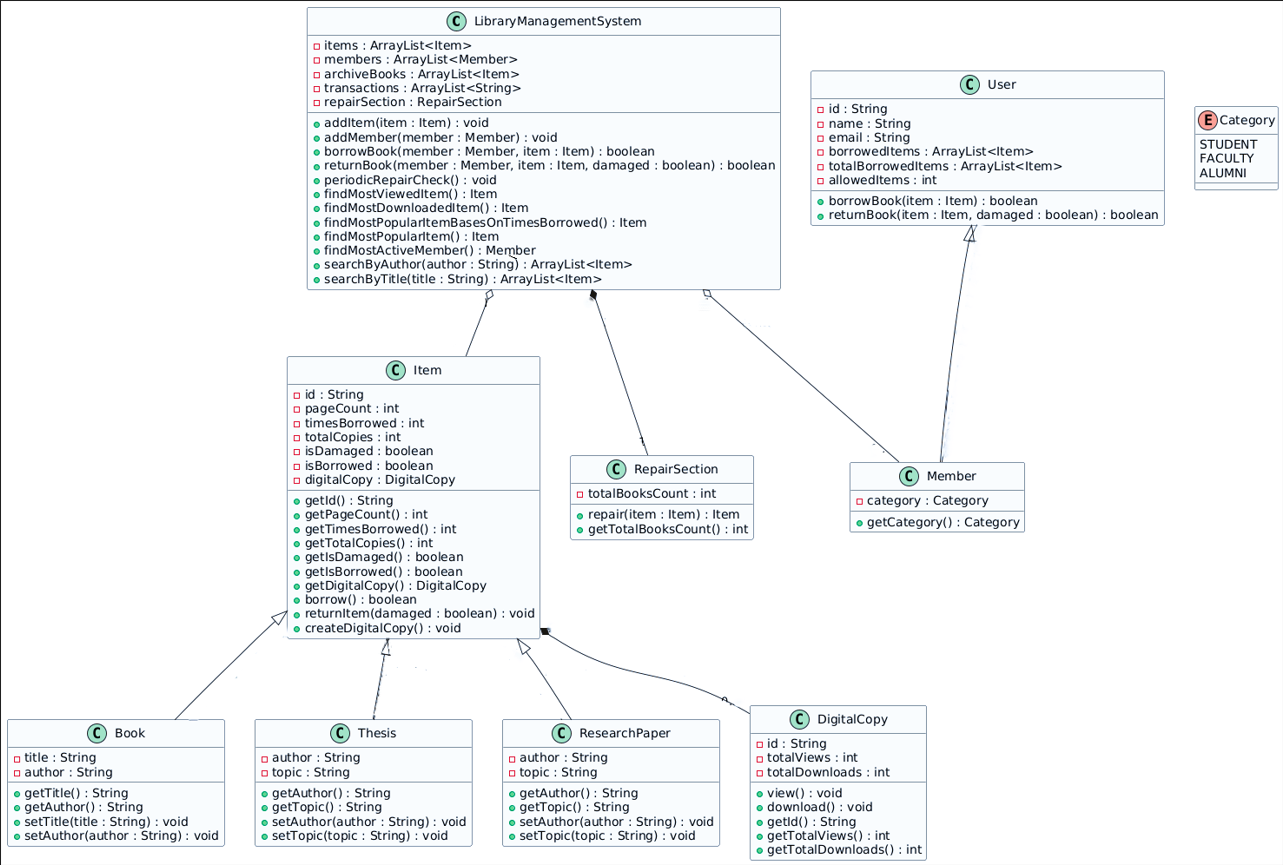
# Neptune Code: H7WPC9

# Name : AHMAD ZOHAIB

# University Library Management System

The university library management system maintains a diverse collection of items and facilitates borrowing and returning processes for members. Items, have an id, a title, an author, page count, topic. The system tracks how many times an item has been borrowed. The item could be one a book, thesis, or research paper. Each item might have multiple copies and new copies can be added. If an item is borrowed, and there are more copies, it should still be available for others to be borrowed; if all copies were borrowed then it would not be available. When returning an item, it would be available for borrowing again if not damaged. A book could get damaged due to misuse and when returning it should be marked as damaged or not, if it’s damaged the item needs repair and a repair must be requested. An item can have a digital copy. By default the item wouldn’t have one, but it can be created. When created, it would have same id as the item. Members can view the item, and if they want they can download the digital copy. The number of views and downloads is kept. The library should be able to find most viewed and most downloaded items. To borrow items, users must be members of the library, they have their own id, name, email, and they have a category of student, faculty, or alumni. Each user keeps an active list of currently borrowed items and initially it should not exceed 3 per member. They should also keep the number of total borrowed items (when a book is successfully returned, then it is added to the list). For each 3 items that are returned, the member’s loyalty would be appreciated by increasing the number of allowed items. However, to be able to consider a return of an item as a loyalty point, the condition of the item should be marked as preserved (not damaged). The Library has a Repairs Section. It keeps count each time an item is sent to be repaired. If damage is severe (use randomness) then the book should be removed from the library to the archives of the library. If successfully repaired then it would normally be back to be available. The Library does a periodical check every now and then to send the items that need repair to repair Section, and to do the repair process. The library stores the collection of items, It also tracks active transactions (borrowing and returning) and provides search functionalities by title and author to help users locate desired materials. The Library can add new items to its collection and manages the borrowing and returning of items. The library also keeps track of all their members, and can search for most active members. Also the people can find most popular items based on the number of times they have been borrowed: the most popular digital item, 2 and the most popular in general taking in account both physical and digital copies. The main simulation must initialize the Library. Create items and add them to the library. Some items might have digital copies. Create members and make borrowing and returning transactions and other necessary operations such as searching for specific items, or making reairs. Read the input from a file, and use as many input files as needed.

# UML Diagram:



# Use Case Diagram:

A diagram of a plane

AI-generated content may be incorrect.

# Pseudo Code:

**LibraryManagementSystem Class**

addItem(item: Item) -> void:

items.add(item)

addMember(member: Member) -> void:

members.add(member)

borrowBook(member: Member, item: Item) -> boolean:

if member in members AND member.borrowBook(item):

transactions.add("Borrowed: " + item.getId() + " by " + member.getId())

return true

return false

returnBook(member: Member, item: Item, damaged: boolean) -> boolean:

if member.returnBook(item, damaged):

transactions.add("Returned: " + item.getId() + " by " + member.getId())

return true

return false

periodicRepairCheck() -> void:

damagedItems = []

for item in items:

if item.ifDamaged():

damagedItems.add(item)

items.remove(item)

for item in damagedItems:

severity = random.nextInt(10)

if severity < 5:

archiveBooks.add(item)

else:

repairSection.repair(item)

items.add(item)

findMostViewedItem() -> Item:

return MAX(items, f(item) => item.getDigitalCopy().getTotalViews())

findMostDownloadedItem() -> Item:

return MAX(items, f(item) => item.getDigitalCopy().getTotalDownloads())

findMostPopularItem() -> Item:

return MAX(items, f(item) => item.getTimesBorrowed() +

(item.getDigitalCopy() ? item.getDigitalCopy().getTotalDownloads() : 0))

findMostActiveMember() -> Member:

return MAX(members, f(member) => member.getTotalBorrowedItems().size())

searchByAuthor(author: String) -> ArrayList<Item>:

result = []

for item in items:

if item.getAuthor() == author:

result.add(item)

return result

searchByTitle(title: String) -> ArrayList<Item>:

result = []

for item in items:

if item instanceof Book AND item.getTitle() == title:

result.add(item)

return result

**User/Member Class**

borrowBook(item: Item) -> boolean:

if borrowedItems.size() < allowedItems AND item.borrow():

borrowedItems.add(item)

return true

return false

returnBook(item: Item, damaged: boolean) -> boolean:

if borrowedItems.remove(item):

item.returnItem(damaged)

if not damaged:

totalBorrowedItems.add(item)

if totalBorrowedItems.size() % 3 == 0:

allowedItems = allowedItems + 1

return true

return false

**Item Class**

borrow() -> boolean:

if totalCopies > 0:

totalCopies = totalCopies - 1

timesBorrowed = timesBorrowed + 1

if totalCopies == 0:

isBorrowed = true

return true

return false

returnItem(damaged: boolean) -> void:

totalCopies = totalCopies + 1

if damaged:

isDamaged = true

if totalCopies > 0:

isBorrowed = false

createDigitalCopy() -> void:

if digitalCopy == null:

digitalCopy = new DigitalCopy(id)

ifDamaged() -> boolean:

return isDamaged

requestRepair() -> boolean:

return isDamaged

ifBorrowed() -> boolean:

return isBorrowed

**DigitalCopy Class**

view() -> void:

totalViews = totalViews + 1

download() -> void:

totalDownloads = totalDownloads + 1

**RepairSection Class**

repair(item: Item) -> Item:

if item.ifDamaged():

totalBooksCount = totalBooksCount + 1

item.setIsDamaged(false)

return item