Project: Developing and Optimizing Data Structures for Real-World Applications Using Python

Name: Kevin Chemutai
Student ID: 005029582
Course Title: Data Structures and Algorithms.

Deliverable 2: Proof of Concept Implementation

GitHub Repo: https://github.com/kchemutai/TextbookManagementSystem

Project: Textbook Management System

Partial Implementation for Textbook Management System (Phase 2: Proof of Concept)

1. Partial Implementation of Data Structures

Core Components Implemented:

- Hash Table for storing and retrieving books by ISBN.
- Stack for tracking recent transactions (checkouts and returns).
- Linked List for managing borrowing history of each book.

Python Implementation:

Book Class

```
from linked_list import LinkedList


class Book:
    def __init__(self, isbn, title, author, edition, category, availability):
        self.isbn = isbn
        self.title = title
        self.author = author
        self.edition = edition
        self.category = category
        self.availability = availability  # Boolean
        self.borrow_history = LinkedList()
```

book_inventory.py

```
from book import Book
import logging

# Configure logging
logging.basicConfig(level=logging.ERROR, format='%(asctime)s - %(levelname)s - %(message)s')

book_inventory = {}

def add_book(isbn, title, author, edition, category):
    book = Book(isbn, title, author, edition, category, True)
    book_inventory[isbn] = book

def retrieve_book(isbn):
    if isbn not in book_inventory:
        logging.error(f"Book with ISBN {isbn} not found in the inventory.")
        raise KeyError(f"Book with ISBN {isbn} not found in the inventory.")
    return book_inventory[isbn]

def update_availability(isbn, availability):
    if isbn not in book_inventory:
        logging.error(f"Cannot update availability: Book with ISBN {isbn} not found.")
        raise KeyError(f"Cannot update availability: Book with ISBN {isbn} not found.")
    book_inventory[isbn].availability = availability
```

Stack for Recent Transactions

transaction.py

```
class Transaction:
    def __init__(self, isbn, user, action, date):
        self.isbn = isbn
        self.user = user
        self.action = action  # 'borrow' or 'return'
        self.date = date
```

recent_transactions.py

```
recent_transactions = []

def add_transaction(transaction):
    recent_transactions.append(transaction)

def get_recent_transaction():
    return recent_transactions.pop() if recent_transactions else None
```

borrow_history.py

```
class BorrowHistoryNode:
    def __init__(self, user, borrow_date):
        self.user = user
        self.borrow_date = borrow_date
        self.return_date = None
        self.next = None
```

Linked List for Borrowing History

linked_list.py

```
from borrow_history import BorrowHistoryNode
```

```python
class LinkedList:
    def __init__(self):
        self.head = None

    def add_borrow_record(self, user, borrow_date):
        new_record = BorrowHistoryNode(user, borrow_date)

        new_record.next = self.head
        self.head = new_record

    def update_return_record(self, user, return_date):
        current = self.head
        while current:
            if current.user == user and current.return_date is None:
                current.return_date = return_date
                break
            current = current.next
```

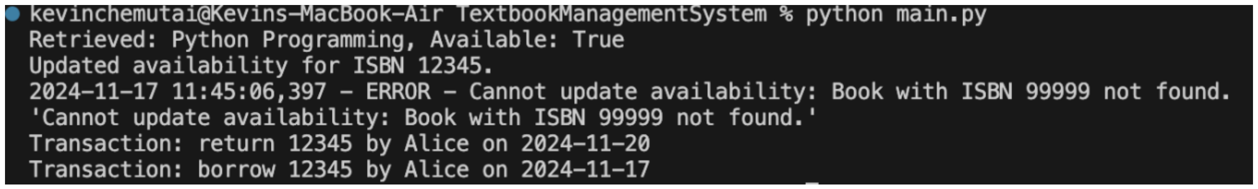
main.py

```python
from book_inventory import add_book, retrieve_book, update_availability
from recent_transactions import add_transaction, get_recent_transaction
from transaction import Transaction

# Adding books
add_book("12345", "Python Programming", "John Doe", "1st", "Technology")
add_book("67890", "Data Structures", "Jane Smith", "2nd", "Computer Science")
```

```python
# Error handling demonstration
try:
    # Retrieving a book
    book = retrieve_book("12345")
    print(f"Retrieved: {book.title}, Available: {book.availability}")
except KeyError as e:
    print(e)

try:
    # Update availability for an existing book
    update_availability("12345", False)
    print(f"Updated availability for ISBN 12345.")
except KeyError as e:
    print(e)

try:
    # Update availability for a non-existent book
    update_availability("99999", False)
except KeyError as e:
    print(e)

# Borrow a book
book = retrieve_book("12345")
book.borrow_history.add_borrow_record("Alice", "2024-11-17")
add_transaction(Transaction("12345", "Alice", "borrow", "2024-11-17"))

# Return a book
book.borrow_history.update_return_record("Alice", "2024-11-20")
update_availability("12345", True)
add_transaction(Transaction("12345", "Alice", "return", "2024-11-20"))

# View recent transactions
while True:
    transaction = get_recent_transaction()
    if transaction is None:
        break
    print(f"Transaction: {transaction.action} {transaction.isbn} by {transaction.user} on {transaction.date}")
```

2. Demonstration of Key Operations

When you run the main.py file, here is the expected output.



```
kevinchemutai@Kevins-MacBook-Air TextbookManagementSystem % python main.py
Retrieved: Python Programming, Available: True
Updated availability for ISBN 12345.
2024-11-17 11:45:06,397 - ERROR - Cannot update availability: Book with ISBN 99999 not found.
'Cannot update availability: Book with ISBN 99999 not found.'
Transaction: return 12345 by Alice on 2024-11-20
Transaction: borrow 12345 by Alice on 2024-11-17
```

3. Documentation of Implementation Process

- **Challenges:** Ensuring proper linkage between borrow and return records in the LinkedList and managing stack overflow for excessive transactions.
- **Solutions:** Implement modular and reusable classes for easier debugging and scalability.
- **Design Changes:** Added transaction tracking via stack for recent actions, which was not explicitly included in the initial design.

4. Code Quality and Best Practices

- Used modular functions and classes.
- Followed Python naming conventions and added meaningful comments.
- Provided error handling for missing ISBNs.

Next Steps

- Integrate a Binary Search Tree for categorizing books.
- Implement advanced search functionality across attributes like category and author.
- Add concurrency control for multi-user environments.

References

1. Knuth, D. E. (1998). *The Art of Computer Programming.* Addison-Wesley.
2. Cormen, T. H., et al. (2009). *Introduction to Algorithms.* MIT Press.

3. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in Python.* Wiley.