

**FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY**

page1image9370432

**PROG211 – Objected Oriented Programming Methods 1**

page1image9370432

Title : Design Rationale for Mini Library Management

Issue Date : Week 2

Due Date : Week 4

Lecturer/Examiner : Mr Amadus Cocker

Name of Student/s : Martin Manyie Gborie

Student ID No. : 905005717

Class : BIT 1104F

Year/Semester : 2/1

Academic Honesty Policy Statement

page1image9370432

I/We, hereby attest those contents of this attachment are my own work. Referenced works, articles, art, programs, papers or parts thereof are acknowledged at the end of this paper. This includes data excerpted from CD-ROMs, the Internet, other private networks, and other people’s disk of the computer system.

Student’s Signature: Date: 15TH OCTOBER 2025

page1image9370432

LECTURER’S COMMMENTS/GRADE:

for office use only upon receive

Remark

DATE:  
 TIME:

RECEIVER’S NAME:

**Design Rationale for Mini Library Management System**

1. Introduction

This document outlines the design choices for the Mini Library Management System developed in Python for Assignment 1 of Object-Oriented Programming 1. The system manages books and members, allowing operations such as adding, updating, deleting, searching, borrowing, and returning books. The rationale focuses on the selection of data structures—dictionaries, lists, and tuples—and justifies their use based on efficiency, readability, and alignment with the assignment requirements.

2. Data Structure Selection

- Dictionary (`books`): A dictionary was chosen to store book details, using ISBN as the key and a tuple of (title, author, genre, total\_copies) as the value. This structure ensures fast lookups (O(1) average time complexity) by ISBN, which is critical for a library system where unique identification is essential. The tuple value maintains immutability for core book attributes, preventing unintended modifications after entry. This choice supports the CRUD operations efficiently, especially for searching and updating specific books.

- List (`members`): A list was selected to manage member records, each represented as a tuple of (member\_id, name, email, borrowed\_books). Lists allow dynamic addition and removal of members, which is necessary as the library grows. The sequential access nature of lists supports iteration over members for borrowing and returning operations, though it may lead to O(n) search time for large datasets. For this small-scale system, this trade-off is acceptable.

- Tuple (`genres`): A tuple was used to define a fixed set of valid genres (Fiction, Non-Fiction, Sci-Fi). Tuples are immutable, ensuring that the genre list cannot be altered accidentally, which enforces data integrity. This choice aligns with the requirement to validate genres during book addition, providing a constant reference point for the system.

3. Justification of Choices

The use of a dictionary for `books` leverages Python’s hash table implementation, making it ideal for quick retrieval and modification based on ISBN. This is particularly useful for the `search\_book` and `update\_book` functions, where efficiency is paramount. The list for `members` supports the dynamic nature of membership, allowing the system to scale as new users join, which is handled by the `add\_member` and `borrow\_book` functions. The tuple for `genres` ensures that only predefined categories are accepted, reducing errors during data entry and maintaining consistency across the system. These structures collectively meet the assignment’s goal of building a functional library system using Python’s built-in data types.

4. Trade-offs and Considerations

While dictionaries offer speed, they require unique keys, which is enforced by checking ISBN duplicates in `add\_book`. This adds a minor overhead but ensures data integrity. Lists, while flexible, may become inefficient for large member bases due to linear search times; however, the assignment’s scope suggests a small system, mitigating this concern. Tuples, though less flexible than lists, provide safety by preventing genre modifications, which is a deliberate design decision to uphold the system’s rules. Future enhancements could involve switching to a class-based structure with methods to encapsulate these data structures, improving modularity.

5. Conclusion

The combination of dictionaries, lists, and tuples creates a robust foundation for the Mini Library Management System. This design balances performance, simplicity, and adherence to OOP principles, fulfilling the assignment’s objectives. The UML diagram (provided as `UML.png`) visually represents this structure, reinforcing the rationale presented here. This approach ensures the system is both functional and maintainable within the given constraints.