

Gabriel Q. Escobido

BSCpE – 2A

SOFTWARE DESIGN

Laboratory Exercise No. 4 CH3

Title: Problem-Solving Strategies: Abstraction and Decomposition

Brief Introduction

Problem-solving is a fundamental skill in software design. This exercise focuses on abstraction and decomposition, two strategies that simplify complex problems by breaking them into manageable components.

Objectives

- Learn how to abstract a problem into its key components.
- Practice decomposing a complex problem into smaller tasks.
- Solve a real-world problem using abstraction and decomposition.

Detailed Discussion

Abstraction involves focusing on the essential aspects of a problem while ignoring irrelevant details. For example, when designing a software system for managing students' grades, you focus on operations like adding, retrieving, and calculating grades rather than the storage mechanism.

Decomposition involves dividing a problem into smaller, manageable tasks. Using the same example, you could decompose it into modules such as:

1. Input module for student data.
2. Calculation module for grades.
3. Output module for reports.

Problem-Solving Strategy	Description	Example
Abstraction	Ignoring irrelevant details.	Focus on grade calculation logic
Decomposition	Breaking problems into smaller parts.	Create input/output modules

Materials

- Python environment
- VS Code IDE

Procedure

1. Identify a real-world problem such as managing a library system.
2. Apply abstraction to focus on core operations:
 - Add a book.
 - Borrow a book.
 - Return a book.
3. Use decomposition to divide the system into smaller tasks such as:
 - Input handling for book details.
 - Logic for borrow/return operations.
 - Output for system status.
4. Implement the following Python program:

Example Code:

Abstraction: Core operations for a library system

class Library:

def __init__(self):

self.books = []

def add_book(self, title):

self.books.append(title)

print(f'Book '{title}' added to the library.")

def borrow_book(self, title):

if title in self.books:

self.books.remove(title)

print(f'You borrowed '{title}'.")

else:

```
print("Book not available.")
```

```
def return_book(self, title):
```

```
    self.books.append(title)
```

```
    print(f'Book '{title}' returned to the library.')
```

Decomposition: Using smaller functions

```
library = Library()
```

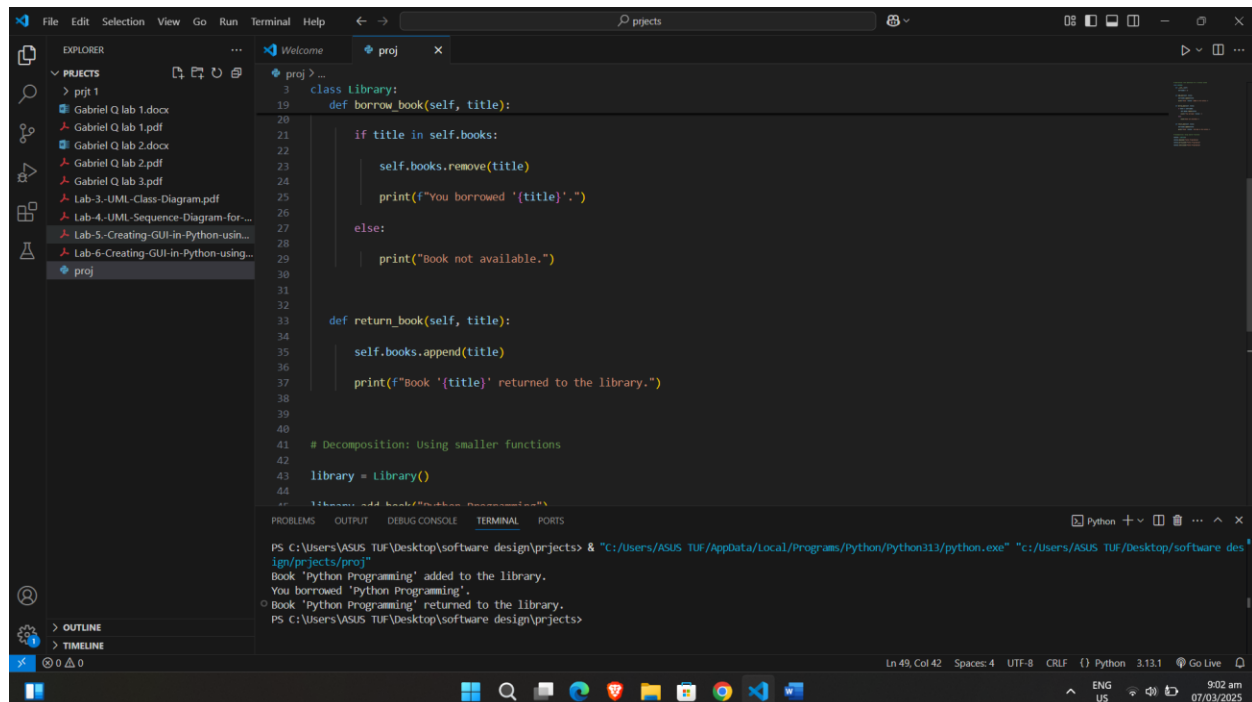
```
library.add_book("Python Programming")
```

```
library.borrow_book("Python Programming")
```

```
library.return_book("Python Programming")
```

Results

Provide screenshots of program execution and explain how abstraction and decomposition were applied in the example.



The screenshot shows a Python IDE with a dark theme. The Explorer panel on the left shows a project named 'proj' with several files. The main editor displays a Python script with the following code:

```
class Library:
    def borrow_book(self, title):
        if title in self.books:
            self.books.remove(title)
            print(f"You borrowed '{title}'.")
        else:
            print("Book not available.")

    def return_book(self, title):
        self.books.append(title)
        print(f"Book '{title}' returned to the library.")

# Decomposition: Using smaller functions
library = Library()

library.add_book("Python Programming")
library.borrow_book("Python Programming")
library.return_book("Python Programming")
```

The terminal at the bottom shows the execution of the program:

```
PS C:\Users\ASUS TUF\Desktop\software design\projects> & "C:/Users/ASUS TUF/AppData/Local/Programs/python/python313/python.exe" "c:/Users/ASUS TUF/Desktop/software design/projects/proj"
Book 'Python Programming' added to the library.
You borrowed 'Python Programming'.
Book 'Python Programming' returned to the library.
PS C:\Users\ASUS TUF\Desktop\software design\projects>
```

The status bar at the bottom indicates the file is at line 49, column 42, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python 3.13.1 interpreter.

Follow-Up Questions

1. What are the benefits of using abstraction in software design?

ANSWER: Abstraction simplifies complex systems by hiding unnecessary details, promoting modularity, and enabling easier updates and reuse of code. It allows developers to focus on functionality instead of implementation, making systems more flexible and maintainable.

2. How does decomposition simplify problem-solving?

ANSWER: Decomposition breaks down a large problem into smaller, manageable pieces, making it easier to understand and solve. It helps focus on specific tasks, improves testing, and makes scaling simpler.

3. Can you identify additional ways to decompose the library system?

ANSWER: You can decompose the library system by separating user roles, catalog management, checkout processes, reservations, notifications, search and recommendations, reporting, database management, admin tasks, and third-party integrations. This structure improves modularity and scalability.