UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 1

Object-oriented Programming

Submitted by:
Sumel, Hendrix Nathan L.

Instructor:
Engr. Maria Rizette H. Sayo

AUGUST 2, 2025

I. Objectives

This laboratory activity aims to implement the principles and techniques in object-oriented programming specifically through:

- Identifying object-orientation design goals
- Identifying the relevance of design pattern to software development

II. Methods

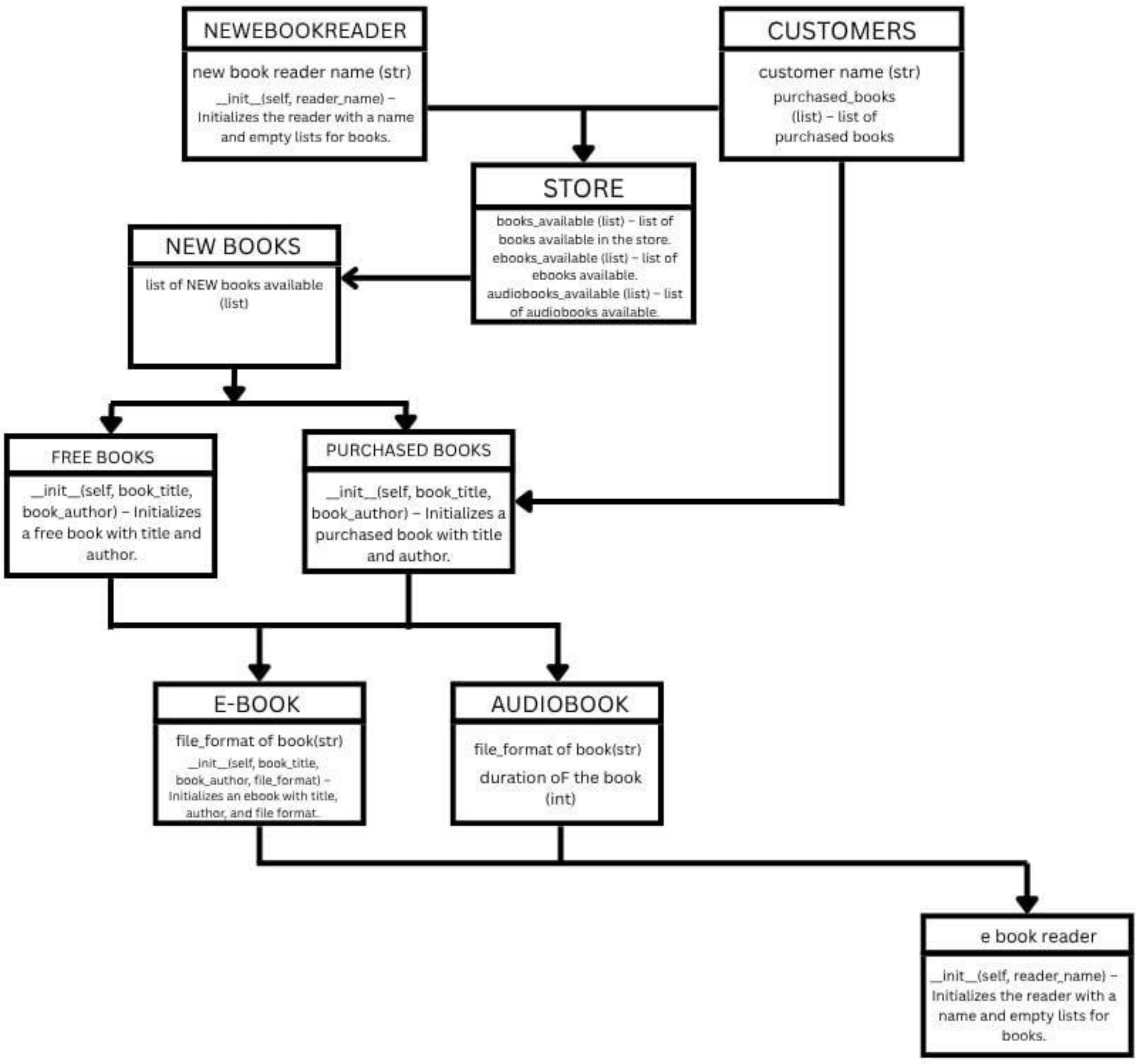
- Software Development
  - The design steps in object-oriented programming
  - Coding style and implementation using Python
  - Testing and Debugging
  - Reinforcement of below exercises

A. Suppose you are on the design team for a new e-book reader. What are the primary classes and methods that the Python software for your reader will need? You should include an inheritance diagram for this code, but you do not need to write any actual code. Your software architecture should at least include ways for customers to buy new books, view their list of purchased books, and read their purchased books.

B. Write a Python class, Polygons that has three instance variables of type str, int, and float, that respectively represent the name of the polygon, its number of sides, and its area. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type and retrieving the value of each type.

III. Results

A.


NEWEBOOKREADER
new book reader name (str)
__init__(self, reader_name) – Initializes the reader with a name and empty lists for books.
CUSTOMERS
customer name (str)
purchased_books (list) – list of purchased books
STORE
books_available (list) – list of books available in the store.
ebooks_available (list) – list of ebooks available.
audiobooks_available (list) – list of audiobooks available.
NEW BOOKS
list of NEW books available (list)
FREE BOOKS
__init__(self, book_title, book_author) – Initializes a free book with title and author.
PURCHASED BOOKS
__init__(self, book_title, book_author) – Initializes a purchased book with title and author.
E-BOOK
file_format of book(str)
__init__(self, book_title, book_author, file_format) – Initializes an ebook with title, author, and file format.
AUDIOBOOK
file_format of book(str)
duration oF the book (int)
e book reader
__init__(self, reader_name) – Initializes the reader with a name and empty lists for books.

So, this flowchart is all about how a new e-book reader system works, broken down into different parts or classes.

First, you got the **NewBookReader** which is basically the interface itself or the device itself. It keeps track of the reader's name and has empty lists ready for books.

Then, there’s the **Customers** part — these are the people who use the reader. Each customer’s got a name, and a list of books they bought. They can buy books, look at their list, and read the books they got.

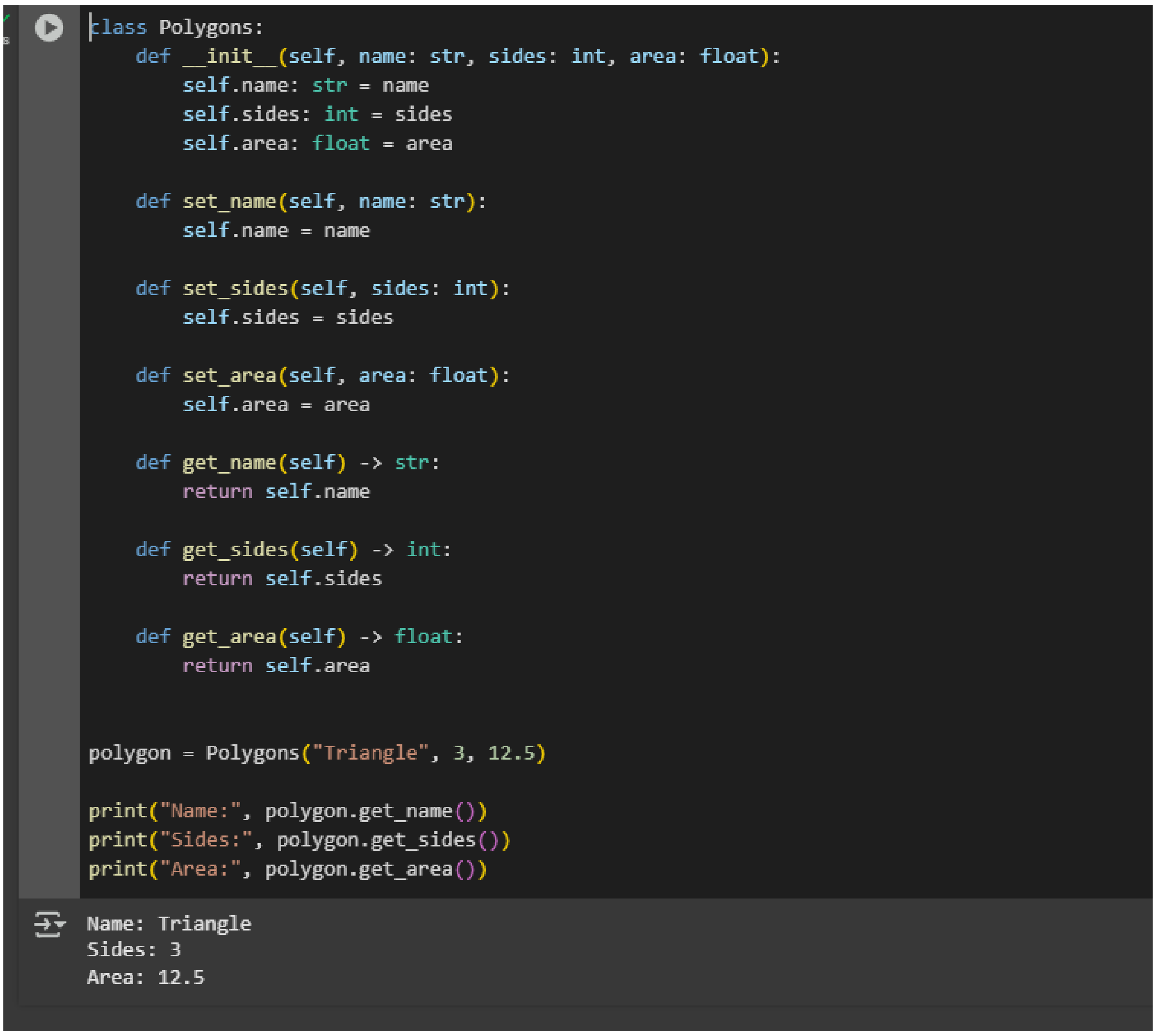
The **Store** is where all the books live. It has lists of new books, e-books, and audiobooks that are available for the customers to buy. So, if you want a book, you check what is available here.

Now, the books themselves are split into different groups. Some books are **Free Books** which you just get without paying. Others are **Purchased Books,** meaning you need to buy them which is within purchased books. there’s also regular **E-Books** and **Audiobooks**. E-books have a file format (like PDF), and audiobooks have that plus how long they are.

The customer picks a book from the store, buys it, and then it gets added to their purchased books list. After that, they can read or listen to it anytime using the reader.

All these parts connect with each other through these classes in the system, making sure everything works smoothly like who owns what book and what books are for sale so basically, this setup makes the software organized, easy to add on to later, and keeps track of customers and books neatly.

B.



```python
class Polygons:
    def __init__(self, name: str, sides: int, area: float):
        self.name: str = name
        self.sides: int = sides
        self.area: float = area

    def set_name(self, name: str):
        self.name = name

    def set_sides(self, sides: int):
        self.sides = sides

    def set_area(self, area: float):
        self.area = area

    def get_name(self) -> str:
        return self.name

    def get_sides(self) -> int:
        return self.sides

    def get_area(self) -> float:
        return self.area


polygon = Polygons("Triangle", 3, 12.5)

print("Name:", polygon.get_name())
print("Sides:", polygon.get_sides())
print("Area:", polygon.get_area())
```

```
Name: Triangle
Sides: 3
Area: 12.5
```

This code sets up a class called Polygons that helps you keep track of polygons by their name, how many sides they have, and their area.

- When you first make a Polygons object, the special __init__ method runs. It takes the name (like "Triangle"), the number of sides (like 3), and the area (like 12.5) and saves them inside the object.
- There are also some handy methods to **change** those values later if you want — like set_name, set_sides, and set_area.
- And if you want to **check** what those values are, you just call the get_name, get_sides, and get_area methods.

IV. Conclusion

In this laboratory activity, I was able to apply the principles of object-oriented programming by identifying design goals, exploring design patterns, and implementing Python classes with constructors, setters, and getters. The exercises allowed me to practice not only the basics of OOP but also to think in terms of real-world applications such as designing an e-book reader system and creating a class for polygons.

Honestly, it was one heck of a job that I did the new book e-reader. It was really hard, especially since it required me to analyze the relationships between classes, think about how users interact with the system, and even design both a flowchart and an inheritance diagram. At first, I struggled because OOP is still new to me and we were only taught the little tto no basics, but I pushed through by using tools like Canva for the diagrams and Google colab for the code.

Overall, this activity reinforced my understanding of object-oriented design and programming. Even though it was challenging, I learned the importance of proper planning, organization of classes, and how design patterns play a role in building maintainable software. The difficulty actually helped me improve, and in the end, I was able to create outputs that connected both theory and practice.

References

[1] Co Arthur O.. "University of Caloocan City Computer Engineering Department Honor Code," UCC-CpE Departmental Policies, 2020.

W3Schools. (2024). *Python Classes and Objects*. https://www.w3schools.com/python/python_classes.asp