Data Structure and Algorithm

Laboratory Activity No. 1

Object-oriented Programming

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Hermosura, Leigh B. | Engr. Maria Rizette H. Sayo |

JULY 28, 2025

# 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

# Methods

* Software Development
  + The design steps in object-oriented programming
  + Coding style and implementation using Python
  + Testing and Debugging
  + Reinforcement of below exercises
  1. 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.
  2. 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.

# Results

The results of the laboratory activity will be presented in this chapter, including graphs, images, and explanation to each part.

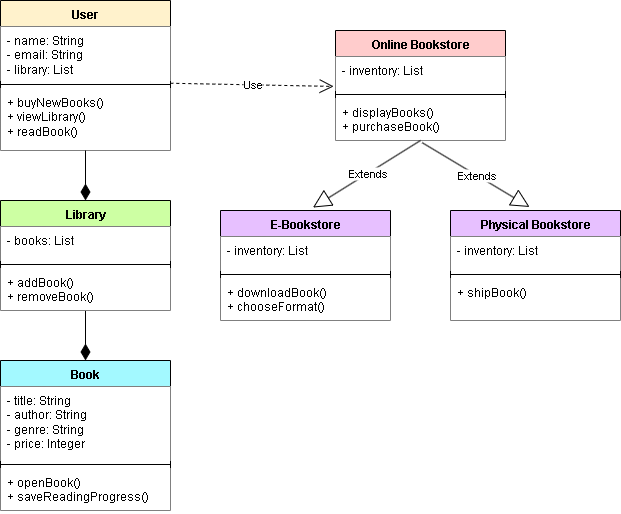


Figure 1 Inheritance diagram of method A

Unified Modeling Language (UML) diagrams visually presents how objects interact inside a system which provide program understanding, debugging, software maintenance, and software archeology [1]. Figure 1 illustrates the framework of the relationship between different classes starting from the parent class.

The superclass *Online Bookstore* manages a list of books and stores them into the *inventory* variable. It includes a *displayBooks()* method to show available titles and a *purchaseBook()* method that allows users to buy books. Subclasses *E-Bookstore* and *Physical Bookstore* inherits the attributes of the *Online Bookstore*. Under *E-Bookstore*, the user can download the digital copy of the book they had bought and select their preferred file format such as PDF, EPUB, and AZW3. *Physical Bookstore* on the other hand has a function that would ship the physical copy of the book to the user.

The class *User* contains attributes of username, email, and a library which stores the user's owned books. The methods of the User class include *buyNewBooks()* to purchase books from the store, *viewLibrary()* to browse books that they had purchased, and *readBook()* to read bought books. The *User* class has a subclass called *Library* that contains their collection of books, having functions to remove or add books. Lastly, the *Library’s* subclass, *Book,* containing attributes such as title, author, genre, and price, along with methods to open the selected book and save the user’s reading progress.

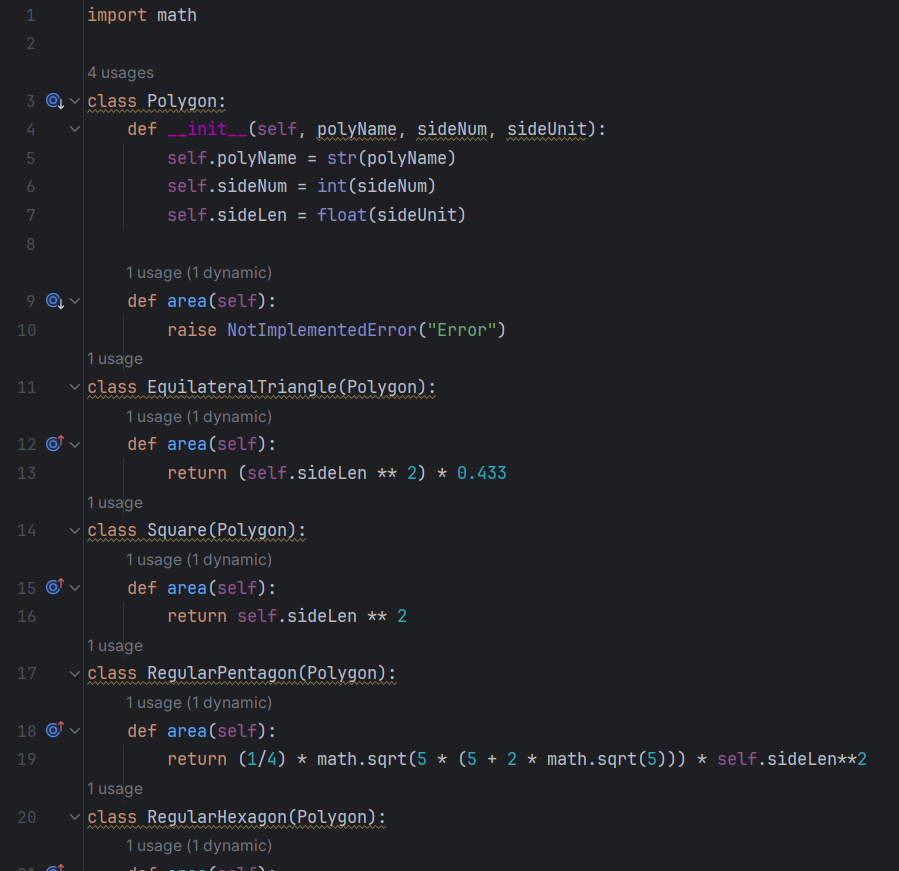


Figure 2 Screenshot of the program for method B

The program imports functions from the math module in order to perform complex operations like radicals. The superclass *Polygon* is defined and under it, the polygon name, side number, and side length are initialized. Inside the superclass is a function for the area, it raises an error when called directly as it’s meant to be overridden by other subclasses.

Beneath the superclass are subclasses such as *EquilateralTriangle*, *Square*, *RegularPentagon*, and *RegularHexagon*, each containing functions that would calculate the area and return its value using formulae unique to each shape.

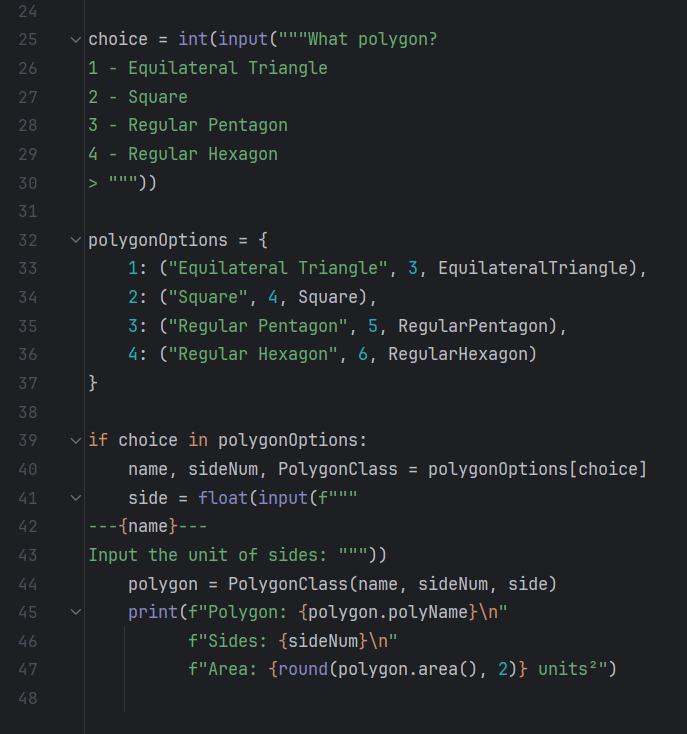


Figure 3 TUI of the program

This section of the program contains the back end for the Text User Interface (TUI). While minimalistic in design, its intention is to engage the user, also allowing flexible inputs instead of a fixed outcome, improving overall user experience [2].

The program begins by asking for the user to choose between four polygon types: equilateral triangle, square, regular pentagon, and regular hexagon. The selected option is stored into the variable *choice*. The next segment is a dictionary named *polygonOptions* where each key corresponds to the numeric choices provided. Each key contains three values: the name of the polygon, number of sides and the class associated with it.

The final part of the code identifies if the user’s choice is found inside the *polygonOptions* dictionary. If valid, it assigns the three values into the variables *name*, *sideNum*, and *PolygonClass*. The program then asks the user to enter the length of a side and calls the appropriate function to calculate the area. Finally, it prints the polygon’s name, number of sides and area, as shown in the output below.

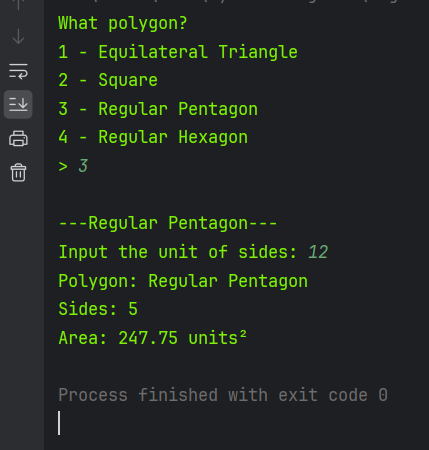


Figure 4 Example output of the program

# Conclusion

The laboratory report contains two main parts: theoretical planning and practical application. The inheritance diagram shows the process of how each segment of code interacts inside a computer system. It is very important in Object-Oriented Programming, as it serves as a basis for programmers to organize their ideas and gain a deeper understanding of their program’s structure and behavior. The second part focuses on the practical application of code, where the programmer creates a working system, incorporating the fundamentals of class inheritance. This validates the initial planning and emphasizes the importance of having a well-structured design in software development.

**References**

[1] C. Alvin, B. Peterson, and S. Mukhopadhyay, “Static generation of UML sequence diagrams,” *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 1, pp. 31–53, 2021, doi: 10.1007/s10009‑019‑00545‑z.

[2] Hotjar, “User Interface (UI),” *Hotjar UI Design Glossary*, https://www.hotjar.com/ui-design/glossary/user-interface/ (accessed Jul. 27, 2025).