Data Structure and Algorithm

Laboratory Activity No. 1

Object-oriented Programming

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Vasig, Yuan Hessed O. | Engr. Maria Rizette H. Sayo |

July,26. 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.

**1. User**

**Attributes:**

* user\_id: str – unique identifier for the user
* name: str
* email: str
* library: list[Book] – list of purchased books (EBook or Audiobook)
* **Operations:**
* buy\_book(book: Book, store: Store) -> None  
  *Purchases a book from the store and adds it to the library.*
* view\_library() -> list[Book]  
  *Returns all purchased books.*
* open\_book(book\_id: str) -> BookReader  
  *Creates a reader instance for the selected book.*

**2. Store**

**Attributes:**

* catalog: list[Book] – available books in the store

**Operations:**

* search\_books(query: str) -> list[Book]  
  *Searches for books by title, author, or keyword.*
* purchase\_book(user: User, book: Book) -> None  
  *Handles purchase logic and adds the book to the user's library.*
* add\_book(book: Book) -> None  
  *Adds a new book to the store catalog.*

**3. Book (Abstract Class)**

**Attributes:**

* book\_id: str
* title: str
* author: str
* price: float

**Operations:**

* get\_metadata() -> dict  
  *Returns metadata such as title, author, price.*

**4. EBook (inherits from Book)**

**Attributes:**

* content: list[str] – list of pages (text)

**5. Audiobook (inherits from Book)**

**Attributes:**

* audio\_files: list[str] – list of audio file paths or URLs
* narrator: str

**6. BookReader**

**Attributes:**

* book: Book
* current\_position: int – page number for EBook or audio index for Audiobook

**Operations:**

* open\_book(book: Book) -> None  
  *Initializes reader for the given book.*
* next() -> None  
  *Moves to next page or audio segment.*
* prev() -> None  
  *Moves to previous page or audio segment.*
* go\_to(position: int) -> None  
  *Jumps to specific page or audio segment.*
* display() -> None  
  *Displays text page or plays audio, depending on book type.*
  1. 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.

Source Code:  
class Polygons:

def \_\_init\_\_(self, name: str, sides: int, area: float):

# Initialize instance variables

self.\_\_name = name

self.\_\_sides = sides

self.\_\_area = area

# Setter methods

def set\_name(self, name: str):

self.\_\_name = name

def set\_sides(self, sides: int):

if sides > 2: # A polygon must have at least 3 sides

self.\_\_sides = sides

else:

raise ValueError("A polygon must have at least 3 sides.")

def set\_area(self, area: float):

if area > 0:

self.\_\_area = area

else:

raise ValueError("Area must be positive.")

# Getter methods

def get\_name(self) -> str:

return self.\_\_name

def get\_sides(self) -> int:

return self.\_\_sides

def get\_area(self) -> float:

return self.\_\_area

# Example usage:

polygon = Polygons("Hexagon", 6, 40)

print("Polygon Name:", polygon.get\_name())

print("Number of Sides:", polygon.get\_sides())

print("Area:", polygon.get\_area())

# Results

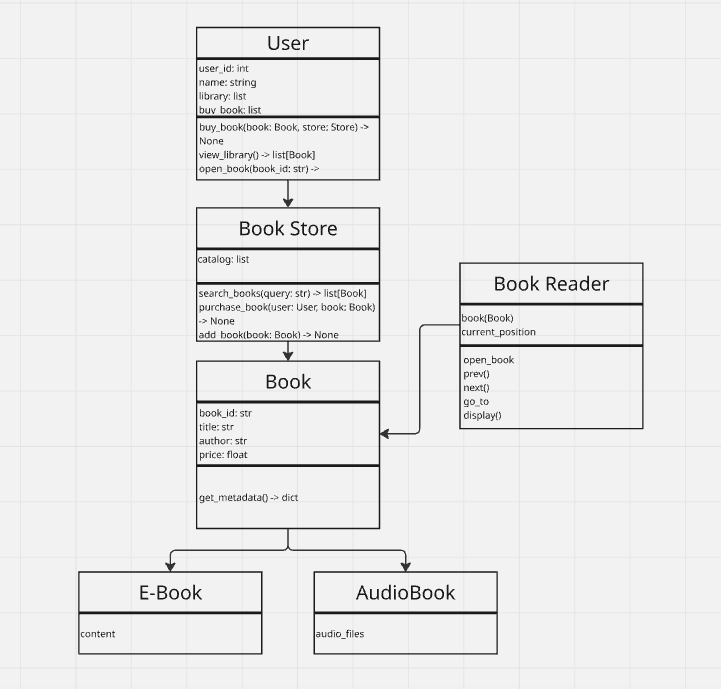


Figure 1 Inheritance diagram

Here shows the relationship of the classes and its attribute in the E-book reader application.

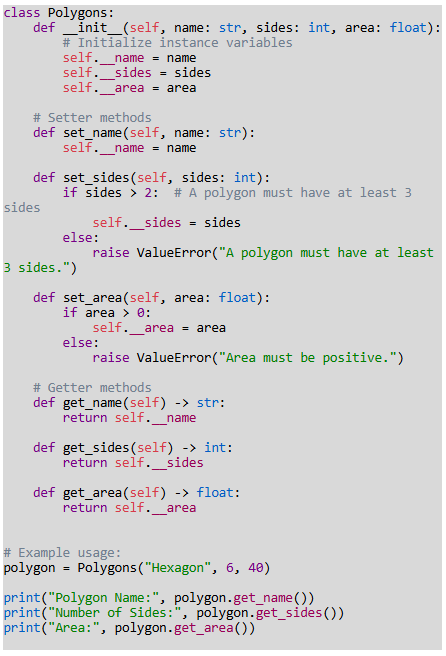


Figure 2 Screenshot of code in w3schools python compiler

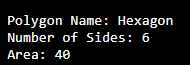


Figure 3: Output of program

The code defines a **Python class Polygons** that represents a polygon. A polygon is a geometric shape with straight sides (like triangle, square, etc.). The class keeps track of:

1. **Name of the polygon** (e.g., "Triangle", "Square")
2. **Number of sides** (e.g., 3 for triangle, 4 for square)
3. **Area** (how much space it covers, as a float)

# Conclusion

I’ve learned how the inheritance functions in an application and how it connects the parent class and the child class throughout the code. Throughout the activity, i’ve had a deeper understanding of coding in python by understanding the concept of inheritance, classes, objects and etc.

**References**

[1] W3Schools, "Python Inheritance," *W3Schools*, [Online]. Available: <https://www.w3schools.com/python/python_inheritance.asp>. [Accessed: Jul. 26, 2025].

[2] Python Software Foundation, "Python Official Website," *Python.org*, [Online]. Available: <https://www.python.org/>. [Accessed: Jul. 26, 2025].

[3] Miro, "Online Collaborative Whiteboard Platform," *Miro*, [Online]. Available: <https://miro.com/>. [Accessed: Jul. 26, 2025].