UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 1

Object-oriented Programming

Submitted by:
Ampong, J-kevin L.

Instructor:
Engr. Maria Rizette H. Sayo

July, 28, 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

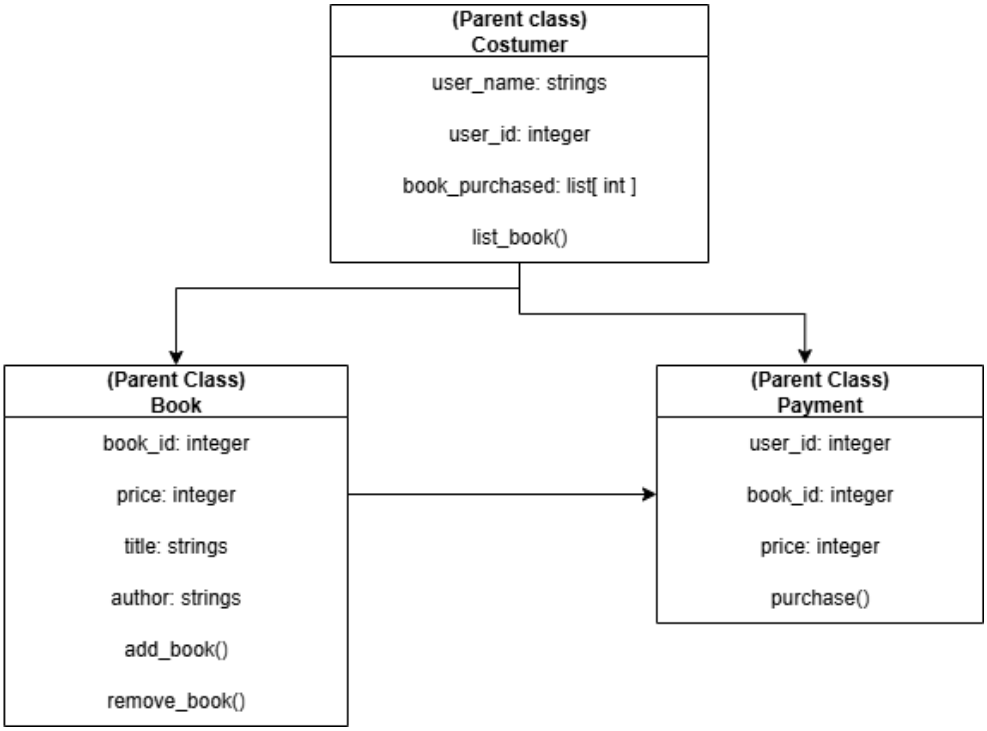
- Software Development
    - The design steps in object-oriented programming
    - Coding style and implementation using Python
    - Testing and Debugging
    - Reinforcement of below exercises

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.



B.

```python
class Polygon:
    polygon_type: str
    sides: int
    area: float

    def __init__(self, sides: int) -> None:
        self.sides = sides

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

    def __str__(self) -> str:
        return f"Type: {self.polygon_type}\n Sides: {self.sides}\n Area: {self.area}"


class Triange(Polygon):
    def __init__(self, sides: int, base: float, height: float) -> None:
        super().__init__(sides)
        self.polygon_type = 'triangle'
        self.area = (1/2) * base * height


class Rectangle(Polygon):
    def __init__(self, sides: int, length: float, width: float) -> None:
        super().__init__(sides)
        self.polygon_type = 'Rectangle'
        self.area = length * width


class Parallelogram(Polygon):
    def __init__(self, sides: int, base: float, height: float) -> None:
        super().__init__(sides)
        self.polygon_type = 'Parallelogram'
        self.area = base * height


triangle = Triange(3, 12, 6)
print(triangle)

rectangle = Rectangle(4, 24, 22)
print(rectangle)

parallelogram = Parallelogram(4, 22, 33)
print(parallelogram)
```

result

```
Type: triangle
 Sides: 3
 Area: 36.0
Type: Rectangle
 Sides: 4
 Area: 528
Type: Parallelogram
 Sides: 4
 Area: 726
```

IV. Conclusion

The conclusion expresses the summary of the whole laboratory report as perceived by the authors of the report.

Classes are a powerful tool for creating reusable code and following the "Don't Repeat Yourself" (DRY) principle. They make it easy to share functionality and extend your codebase without writing a lot of repetitive code. This is especially useful for large projects that require frequent modifications to their business logic.

However, classes aren't always the best solution. For small or simple projects, using a class might lead to writing more code than necessary. In those situations, a different approach, such as using **composition**, might be a better choice.

References

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