Assignment 5

|  |  |  |
| --- | --- | --- |
| **5** | | |
| **Aim:** Create a Shape superclass with a method to calculate the area, and Circle and Rectangle subclasses that override the area calculation method to demonstrate polymorphism.  Create a Book class with private attributes for title, author, and ISBN, and public getter and setter methods to access and modify these attributes, demonstrating encapsulation. | | |
| **Code:**  import math  *# Shape superclass to demonstrate polymorphism*  class Shape:  def \_\_init\_\_(self, name="Shape"):  self.name = name    def calculate\_area(self):  *"""Base method to calculate area of a shape"""*  return 0    def \_\_str\_\_(self):  return f"{self.name} with area: {self.calculate\_area()}"  *# Circle subclass*  class Circle(Shape):  def \_\_init\_\_(self, radius):  super().\_\_init\_\_("Circle")  self.radius = radius    def calculate\_area(self):  *"""Override area calculation for circle"""*  return math.pi \* self.radius \*\* 2  *# Rectangle subclass*  class Rectangle(Shape):  def \_\_init\_\_(self, length, width):  super().\_\_init\_\_("Rectangle")  self.length = length  self.width = width    def calculate\_area(self):  *"""Override area calculation for rectangle"""*  return self.length \* self.width  *# Book class to demonstrate encapsulation*  class Book:  def \_\_init\_\_(self, title, author, isbn):  *# Private attributes using double underscore*  self.\_\_title = title  self.\_\_author = author  self.\_\_isbn = isbn    *# Getter methods*  def get\_title(self):  return self.\_\_title    def get\_author(self):  return self.\_\_author    def get\_isbn(self):  return self.\_\_isbn    *# Setter methods*  def set\_title(self, title):  self.\_\_title = title    def set\_author(self, author):  self.\_\_author = author    def set\_isbn(self, isbn):  if len(isbn) == 13 or len(isbn) == 10: *# Basic validation for ISBN*  self.\_\_isbn = isbn  else:  print("Invalid ISBN format")    def \_\_str\_\_(self):  return f"Book: {self.\_\_title} by {self.\_\_author}, ISBN: {self.\_\_isbn}"  *# Test code to demonstrate the classes*  if \_\_name\_\_ == "\_\_main\_\_":  *# Polymorphism demonstration*  print("POLYMORPHISM DEMONSTRATION")  shapes = [Circle(5), Rectangle(4, 6), Shape()]    for shape in shapes:  print(shape)    *# Encapsulation demonstration*  print("\nENCAPSULATION DEMONSTRATION")  book = Book("Python Programming", "John Smith", "9781234567890")  print(book)    *# Accessing private attributes through getter methods*  print(f"Title: {book.get\_title()}")  print(f"Author: {book.get\_author()}")  print(f"ISBN: {book.get\_isbn()}")    *# Modifying private attributes through setter methods*  book.set\_title("Advanced Python Programming")  book.set\_author("Jane Doe")  book.set\_isbn("1234567891012")  print("\nAfter modification:")  print(book)  **Output Screenshot:** | | |
| **Conclusion/Summary:**  This assignment demonstrates two fundamental principles of object-oriented programming:  Polymorphism: The Shape hierarchy shows how different subclasses (Circle and Rectangle) can implement the same method (calculate\_area()) in different ways while maintaining a consistent interface. This allows for flexible code that can work with various shapes without needing to know their specific implementations.  Encapsulation: The Book class illustrates how to hide implementation details by making attributes private and providing controlled access through getter and setter methods. This protects the data integrity (as shown by the ISBN validation) and creates a clean, stable interface for other code to interact with. | | |
| **Student Signature & Date** | **Marks:** | **Evaluator Signature & Date** |