1. Write a python program to create a base class "Shape" with methods to calculate area and perimeter. Then, create derived classes "Circle" and "Rectangle" that inherit from the base class and calculate their respective areas and perimeters. Demonstrate their usage in a program.

You are developing an online quiz application where users can take quizzes on various topics and receive scores.

1. Create a class for quizzes and questions.
2. Implement a scoring system that calculates the user's score on a quiz.
3. How would you store and retrieve user progress, including quiz history and scores?

```
import math

class Shape:
    def calculate_area(self):
        pass

    def calculate_perimeter(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius * self.radius

    def calculate_perimeter(self):
        return 2 * math.pi * self.radius


class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        return self.length * self.width
```

```
    def calculate_perimeter(self):
        return 2 * (self.length + self.width)


circle = Circle(5)
print(f"Circle Area: {circle.calculate_area()}")
print(f"Circle Perimeter: {circle.calculate_perimeter()}")

rectangle = Rectangle(4, 6)
print(f"Rectangle Area: {rectangle.calculate_area()}")
print(f"Rectangle Perimeter: {rectangle.calculate_perimeter()}")


class Question:
    def __init__(self, text, answer):
        self.text = text
        self.answer = answer

class Quiz:
    def __init__(self, name):
        self.name = name
        self.questions = []

    def add_question(self, question):
        self.questions.append(question)


class QuizTaker:
    def __init__(self, name):
        self.name = name
        self.scores = {}

    def take_quiz(self, quiz):
        score = 0
        for question in quiz.questions:
            user_answer = input(question.text + " ")
            if user_answer == question.answer:
                score += 1
        self.scores[quiz.name] = score
        print(f"{self.name}'s score on {quiz.name}: {score}/{len(quiz.questions)}")


quiz1 = Quiz("Math Quiz")
quiz1.add_question(Question("What is 2 + 2?", "4"))
quiz1.add_question(Question("What is 3 * 5?", "15"))

quiz2 = Quiz("Science Quiz")
quiz2.add_question(Question("What is the chemical symbol for oxygen?", "O"))
quiz2.add_question(Question("What is the largest planet in the solar system?", "Jupiter"))
```

```
user1 = QuizTaker("Ravi")
user2 = QuizTaker("suresh")

user1.take_quiz(quiz1)
user2.take_quiz(quiz1)

user1.take_quiz(quiz2)
user2.take_quiz(quiz2)
```

```
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Rectangle Area: 24
Rectangle Perimeter: 20
What is 2 + 2? 4
What is 3 * 5? 15
Ravi's score on Math Quiz: 2/2
What is 2 + 2? 4
What is 3 * 5? 15
suresh's score on Math Quiz: 2/2
What is the chemical symbol for oxygen? O
What is the largest planet in the solar system? jupiter
Ravi's score on Science Quiz: 1/2
What is the chemical symbol for oxygen? O
What is the largest planet in the solar system? Jupiter
suresh's score on Science Quiz: 2/2
```

2. Write a python script to create a class "Person" with private attributes for age and name. Implement a method to calculate a person's eligibility for voting based on their age. Ensure that age cannot be accessed directly but only through a getter method.

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def get_age(self):
        return self.__age

    def is_eligible_to_vote(self):
        if self.__age >= 18:
            return True
```

```
        else:
            return False

person1 = Person("Alice", 25)
print(f"{person1._Person__name}'s age is {person1.get_age()}")


if person1.is_eligible_to_vote():
    print(f"{person1._Person__name} is eligible to vote.")
else:
    print(f"{person1._Person__name} is not eligible to vote.")
```

```
Alice's age is 25
Alice is eligible to vote.
```

Double-click (or enter) to edit

3. You are tasked with designing a Python class hierarchy for a simple banking system. The system should be able to handle different types of accounts, such as Savings Accounts and Checking Accounts. Both account types should have common attributes like an account number, account holder's name, and balance. However, Savings Accounts should have an additional attribute for interest rate, while Checking Accounts should have an attribute for overdraft limit.

```
class BankAccount:

    def __init__(self, account_number, account_holder_name, balance):
        self._account_number = account_number
        self._account_holder_name = account_holder_name
        self._balance = balance

    def get_account_number(self):
```

```
        return self._account_number

    def get_account_holder_name(self):
        return self._account_holder_name

    def get_balance(self):
        return self._balance

    def deposit(self, amount):
        self._balance += amount

    def withdraw(self, amount):
        if amount > self._balance:
            raise ValueError("Insufficient balance")
        self._balance -= amount


class SavingsAccount(BankAccount):

    def __init__(self, account_number, account_holder_name, balance, interest_rate):
        super().__init__(account_number, account_holder_name, balance)
        self._interest_rate = interest_rate

    def get_interest_rate(self):
        return self._interest_rate

    def calculate_interest(self):
        interest = self._balance * self._interest_rate
        self._balance += interest


class CheckingAccount(BankAccount):
    """Class for checking accounts."""

    def __init__(self, account_number, account_holder_name, balance, overdraft_limit):
        super().__init__(account_number, account_holder_name, balance)
        self._overdraft_limit = overdraft_limit

    def get_overdraft_limit(self):
        """Returns the overdraft limit."""
        return self._overdraft_limit

    def withdraw(self, amount):

        if amount > self._balance + self._overdraft_limit:
            raise ValueError("Insufficient balance")
        self._balance -= amount



savings_account = SavingsAccount(12345, "Alice", 1000, 0.03)
```

```
savings_account = SavingsAccount(12345, "Alice", 1000, 0.03)
checking_account = CheckingAccount(54321, "Bob", 500, 200)

savings_account.deposit(500)
checking_account.withdraw(100)
savings_account.calculate_interest()
print(savings_account.get_balance())
print(checking_account.get_balance())
```

```
1545.0
400
```

4. You are developing an employee management system for a company. Ensure that the system utilizes encapsulation and polymorphism to handle different types of employees. such as full-time and part-time employees.

1. Create a base class called "Employee" with private attributes for name, employee ID, and salary. Implement getter and setter methods for these attributes.
2. Design two subclasses, "FullTimeEmployee" and "PartTimeEmployee," that inherit from "Employee." These subclasses should encapsulate specific properties like hours worked (for part-time employees) and annual salary (for full-time employees).
3. Override the salary calculation method in both subclasses to account for different payment structures.
4. Write a program that demonstrates polymorphism by creating instances of both "Full TimeEmployee" and "PartTimeEmployee." Calculate their salaries and display employee information.

```
class Employee:
    def __init__(self, name, employee_id):
        self.__name = name
        self.__employee_id = employee_id
        self.__salary = 0

    def get_name(self):
        return self.__name

    def get_employee_id(self):
```

```
        return self.__employee_id

    def get_salary(self):
        return self.__salary

    def set_salary(self, salary):
        self.__salary = salary

    def calculate_salary(self):
        pass

class FullTimeEmployee(Employee):
    def __init__(self, name, employee_id, annual_salary):
        super().__init__(name, employee_id)
        self.__annual_salary = annual_salary

    def calculate_salary(self):
        self.set_salary(self.__annual_salary)

class PartTimeEmployee(Employee):
    def __init__(self, name, employee_id, hours_worked, hourly_rate):
        super().__init__(name, employee_id)
        self.__hours_worked = hours_worked
        self.__hourly_rate = hourly_rate

    def calculate_salary(self):
        self.set_salary(self.__hours_worked * self.__hourly_rate)


def display_employee_info(employee):
    print(f"Employee Name: {employee.get_name()}")
    print(f"Employee ID: {employee.get_employee_id()}")
    print(f"Salary: ${employee.get_salary()}")


if __name__ == "__main__":
    full_time_employee = FullTimeEmployee("Alice", 101, 60000)
    part_time_employee = PartTimeEmployee("Bob", 102, 30, 20)


    full_time_employee.calculate_salary()
    part_time_employee.calculate_salary()

    print("Full-Time Employee Information:")
    display_employee_info(full_time_employee)
    print("\nPart-Time Employee Information:")
    display_employee_info(part_time_employee)
```

```
 Full-Time Employee Information:
 Employee Name: Alice
```

```
Employee Name: Alice
Employee ID: 101
Salary: $60000

Part-Time Employee Information:
Employee Name: Bob
Employee ID: 102
Salary: $600
```

5. Library Management System-Scenario: You are developing a library management system where you need to handle books, patrons, and library transactions.

1. Create a class hierarchy that includes classes for books (e.g., Book), patrons (e.g.. Patron), and transactions (e.g., Transaction). Define attributes and methods for each class.
2. Implement encapsulation by making relevant attributes private and providing getter and setter methods where necessary.
3. Use inheritance to represent different types of books (e.g., fiction, non-fiction) as subclasses of the Book class. Ensure that each book type can have specific attributes and methods.
4. Demonstrate polymorphism by allowing patrons to check out and return books. regardless of the book type.
5. Implement a method for tracking overdue books and notifying patrons.
6. Consider scenarios like book reservations, late fees, and library staff interactions in your design.

```
from datetime import datetime, timedelta

class Book:
    def __init__(self, title, author, publication_date):
        self.__title = title
        self.__author = author
        self.__publication_date = publication_date
        self.__is_checked_out = False

    def get_title(self):
        return self.__title

    def get_author(self):
        return self.__author
```

```
    def get_publication_date(self):
        return self.__publication_date

    def is_checked_out(self):
        return self.__is_checked_out

    def check_out(self):
        self.__is_checked_out = True

    def return_book(self):
        self.__is_checked_out = False

    def __str__(self):
        return f"{self.__title} by {self.__author}"

class Patron:
    def __init__(self, name, patron_id):
        self.__name = name
        self.__patron_id = patron_id
        self.__checked_out_books = []

    def get_name(self):
        return self.__name

    def get_patron_id(self):
        return self.__patron_id

    def get_checked_out_books(self):
        return self.__checked_out_books

    def check_out_book(self, book):
        if not book.is_checked_out():
            book.check_out()
            self.__checked_out_books.append(book)

    def return_book(self, book):
        if book in self.__checked_out_books:
            book.return_book()
            self.__checked_out_books.remove(book)

    def __str__(self):
        return f"{self.__name} (ID: {self.__patron_id})"

class Transaction:
    def __init__(self, book, patron):
        self.__book = book
        self.__patron = patron
        self.__checkout_date = datetime.now()
        self.__due_date = self.__checkout_date + timedelta(days=14)
```

```
    def get_book(self):
        return self.__book

    def get_patron(self):
        return self.__patron

    def get_checkout_date(self):
        return self.__checkout_date

    def get_due_date(self):
        return self.__due_date

    def is_overdue(self):
        return datetime.now() > self.__due_date

book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "1925")
book2 = Book("To Kill a Mockingbird", "Harper Lee", "1960")
patron1 = Patron("Alice", "P001")
patron2 = Patron("Bob", "P002")

patron1.check_out_book(book1)
patron2.check_out_book(book2)

transaction1 = Transaction(book1, patron1)
transaction2 = Transaction(book2, patron2)

if transaction1.is_overdue():
    print(f"{patron1.get_name()}'s book '{book1.get_title()}' is overdue!")

if transaction2.is_overdue():
    print(f"{patron2.get_name()}'s book '{book2.get_title()}' is overdue!")

patron1.return_book(book1)
patron2.return_book(book2)
```

6.Online Shopping Cart

Scenario: You are tasked with designing a class hierarchy for an online shopping cart system. The system should handle products, shopping carts, and orders. Consider various OOP principles while designing this system.

1. Create a class hierarchy that includes classes for products (e.g., Product), shopping carts (e.g., ShoppingCart), and orders (e.g., Order). Define attributes and methods for each class.
2. Implement encapsulation by making relevant attributes private and providing getter and setter methods where necessary.

3. Use inheritance to represent different types of products (e.g., electronics, clothing) as subclasses of the Product class. Ensure that each product type can have specific attributes and methods.
4. Demonstrate polymorphism by allowing various product types to be added to a shopping cart and calculate the total cost of items in the cart.
5. Implement a method for placing an order, which transfers items from the shopping cart to an order.

Consider scenarios like out-of-stock products, discounts, and shipping costs in your design.

```
class Product:
    def __init__(self, product_id, name, price):
        self.__product_id = product_id
        self.__name = name
        self.__price = price

    def get_product_id(self):
        return self.__product_id

    def get_name(self):
        return self.__name

    def get_price(self):
        return self.__price

    def __str__(self):
        return f"{self.__name} - ${self.__price:.2f}"

class Electronics(Product):
    def __init__(self, product_id, name, price, warranty_period):
        super().__init__(product_id, name, price)
        self.__warranty_period = warranty_period

    def get_warranty_period(self):
        return self.__warranty_period

    def __str__(self):
        return f"{super().__str__()} - Warranty: {self.__warranty_period} months"

class Clothing(Product):
    def __init__(self, product_id, name, price, size):
        super().__init__(product_id, name, price)
        self.__size = size

    def get_size(self):
        return self.__size
```

```
    def __str__(self):
        return f"{super().__str__()} - Size: {self.__size}"

class ShoppingCart:
    def __init__(self):
        self.__items = []

    def add_item(self, product, quantity):
        self.__items.append((product, quantity))

    def remove_item(self, product, quantity):
        if (product, quantity) in self.__items:
            self.__items.remove((product, quantity))

    def calculate_total_cost(self):
        total_cost = 0
        for product, quantity in self.__items:
            total_cost += product.get_price() * quantity
        return total_cost

    def __str__(self):
        cart_contents = "\n".join([f"{product} x{quantity}" for product, quantity in self._
        return f"Shopping Cart:\n{cart_contents}\nTotal Cost: ${self.calculate_total_cost()

class Order:
    def __init__(self, order_id, cart):
        self.__order_id = order_id
        self.__cart = cart
        self.__order_date = None
        self.__is_placed = False

    def place_order(self):
        if not self.__is_placed:
            self.__order_date = "2023-09-18"
            self.__is_placed = True
        else:
            print("This order has already been placed.")

    def __str__(self):
        if self.__is_placed:
            return f"Order ID: {self.__order_id}\nOrder Date: {self.__order_date}\n{self._
        else:
            return f"Order ID: {self.__order_id}\nOrder not placed"


laptop = Electronics("E101", "Laptop", 800, 12)
shirt = Clothing("C201", "T-Shirt", 20, "L")

cart = ShoppingCart()
cart.add_item(laptop, 2)
```

```
cart.add_item(shirt, 3)

order = Order("O001", cart)

order.place_order()

print(order)
```

```
     Order ID: O001
     Order Date: 2023-09-18
     Shopping Cart:
     Laptop - $800.00 - Warranty: 12 months x2
     T-Shirt - $20.00 - Size: L x3
     Total Cost: $1660.00
```