Q2: Student Management System

Define the `Student` class with encapsulated attributes

```
class Student:
    def __init__(self, name, age, roll_number):
        self.__name = name
        self.__age = age
        self.__roll_number = roll_number
    # Getter methods
    def get_name(self):
        return self.__name
    def get_age(self):
        return self.__age
    def get_roll_number(self):
        return self.__roll_number
    # Setter methods
    def set_name(self, name):
        self.__name = name
    def set_age(self, age):
        self.__age = age
    def set_roll_number(self, roll_number):
        self.__roll_number = roll_number
    # Method to display student information
    def display_info(self):
        print(f"Name: {self.__name}, Age: {self.__age}, Roll Number:
{self.__roll_number}")
    # Method to update student details
    def update_details(self, name=None, age=None, roll_number=None):
        if name:
            self.set_name(name)
        if age:
            self.set_age(age)
        if roll_number:
            self.set_roll_number(roll_number)
```

Test the `Student` class

```
# Create instances of the Student class
student1 = Student("Alice", 20, "S123")
student2 = Student("Bob", 22, "S124")
```

```
# Display student information
student1.display_info()
student2.display_info()

# Update student details
student1.update_details(name="Alicia", age=21)
student1.display_info()
```

Q3: Library Resource Management

Define the `LibraryBook` class with encapsulated attributes

```
class LibraryBook:
    def __init__(self, book_name, author, available=True):
        self.__book_name = book_name
        self.__author = author
        self.__available = available

    # Getter methods
    def get_book_name(self):
        return self.__book_name

    def get_author(self):
        return self.__author

    def is_available(self):
        return self.__available

    # Methods for borrowing and returning books
    def borrow_book(self):
        if self.__available:
            self.__available = False
            print(f"You have borrowed '{self.__book_name}'.")
        else:
            print(f"Sorry, '{self.__book_name}' is currently
unavailable.")

    def return_book(self):
        if not self.__available:
            self.__available = True
            print(f"You have returned '{self.__book_name}'.")
        else:
            print(f"'{self.__book_name}' was not borrowed.")
```

Test the `LibraryBook` class

```
# Create instances of the LibraryBook class
book1 = LibraryBook("1984", "George Orwell")
book2 = LibraryBook("To Kill a Mockingbird", "Harper Lee")

# Test borrowing and returning books
book1.borrow_book()
book1.borrow_book()  # Trying to borrow again should show
unavailable
book1.return_book()
book1.return_book()  # Trying to return again should show not
borrowed
```

Q4: Simple Banking System

Define base class and subclasses for different account types

```
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.__account_number = account_number
        self.__balance = balance

    # Getter methods
    def get_account_number(self):
        return self.__account_number

    def get_balance(self):
        return self.__balance

    # Methods for deposit, withdraw, and balance inquiry
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: {amount}, New Balance:
{self.__balance}")
        else:
            print("Invalid deposit amount.")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew: {amount}, New Balance:
{self.__balance}")
        else:
            print("Invalid or insufficient funds for withdrawal.")

    def balance_inquiry(self):
        print(f"Account Number: {self.__account_number}, Balance:
{self.__balance}")

class SavingsAccount(BankAccount):
    def __init__(self, account_number, balance=0,
interest_rate=0.01):
        super().__init__(account_number, balance)
        self.__interest_rate = interest_rate

class CheckingAccount(BankAccount):
    def __init__(self, account_number, balance=0,
overdraft_limit=500):
        super().__init__(account_number, balance)
        self.__overdraft_limit = overdraft_limit
```

Test the banking system

```
# Create instances of different account types
savings_account = SavingsAccount("SA123", 1000)
checking_account = CheckingAccount("CA123", 500)

# Perform transactions
savings_account.deposit(500)
savings_account.withdraw(300)
savings_account.balance_inquiry()

checking_account.deposit(1000)
checking_account.withdraw(1200)
checking_account.balance_inquiry()
```

Q5: Animal Sound System

Define the `Animal` class and its subclasses

```
class Animal:
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        print("Woof!")

class Cat(Animal):
    def make_sound(self):
        print("Meow!")
```

Test the `Animal` subclasses

```
# Create instances of Dog and Cat and call make_sound method
dog = Dog()
cat = Cat()

dog.make_sound()  # Output: Woof!
cat.make_sound()  # Output: Meow!
```

Q6: Restaurant Management System

Define the `MenuItem` class and its subclasses

```
class MenuItem:
    def __init__(self, item_id, name, description, price, category):
        self.__item_id = item_id
        self.name = name
        self.description = description
        self.price = price
        self.category = category

    def get_item_id(self):
        return self.__item_id

    def update_item(self, name=None, description=None, price=None,
category=None):
        if name:
            self.name = name
        if description:
            self.description = description
        if price:
            self.price = price
        if category:
            self.category = category

    def display_info(self):
        print(f"ID: {self.__item_id}, Name: {self.name},
Description: {self.description}, Price: {self.price}, Category:
{self.category}")

class FoodItem(MenuItem):
    def __init__(self, item_id, name, description, price, category,
cuisine_type):
        super().__init__(item_id, name, description, price,
category)
        self.cuisine_type = cuisine_type

    def display_info(self):
        super().display_info()
        print(f"Cuisine Type: {self.cuisine_type}")

class BeverageItem(MenuItem):
    def __init__(self, item_id, name, description, price, category,
is_alcoholic):
        super().__init__(item_id, name, description, price,
category)
        self.is_alcoholic = is_alcoholic

    def display_info(self):
        super().display_info()
        print(f"Alcoholic: {self.is_alcoholic}")

class Menu:
    def __init__(self):
        self.items = {}

    def add_item(self, item):
        self.items[item.get_item_id()] = item
    def remove_item(self, item_id):
        if item_id in self.items:
            del self.items[item_id]

    def update_item(self, item_id, **kwargs):
        if item_id in self.items:
            self.items[item_id].update_item(**kwargs)

    def display_menu(self):
        for item in self.items.values():
            item.display_info()

# Testing the Restaurant Management System
menu = Menu()
food1 = FoodItem(1, "Pizza", "Cheesy Pizza", 12.99, "Main Course",
"Italian")
beverage1 = BeverageItem(2, "Coke", "Chilled Coke", 1.99,
"Beverage", False)

menu.add_item(food1)
menu.add_item(beverage1)
menu.display_menu()

menu.update_item(1, price=13.99)
menu.remove_item(2)
menu.display_menu()
```

Q7: Hotel Management System

Define the Room class and its subclasses

```
class Room:
    def __init__(self, room_id, room_number, room_type, rate,
available=True):
        self.__room_id = room_id
        self.room_number = room_number
        self.room_type = room_type
        self.rate = rate
        self.available = available

    def get_room_id(self):
        return self.__room_id

    def book_room(self):
        if self.available:
            self.available = False
            print(f"Room {self.room_number} has been booked.")
        else:
            print(f"Room {self.room_number} is already booked.")

    def check_in(self):
        if not self.available:
            print(f"Guest checked into room {self.room_number}.")
        else:
            print(f"Room {self.room_number} is available for
booking.")

    def check_out(self):
        if not self.available:
            self.available = True
            print(f"Room {self.room_number} is now available.")
        else:
            print(f"Room {self.room_number} was not occupied.")

class SuiteRoom(Room):
    def __init__(self, room_id, room_number, rate, available=True,
suite_level="Deluxe"):
        super().__init__(room_id, room_number, "Suite", rate,
available)
        self.suite_level = suite_level

class StandardRoom(Room):
    def __init__(self, room_id, room_number, rate, available=True):
        super().__init__(room_id, room_number, "Standard", rate,
available)

# Testing the Hotel Management System
suite_room = SuiteRoom(1, "101", 200.00)
standard_room = StandardRoom(2, "102", 100.00)

suite_room.book_room()
suite_room.check_in()
suite_room.check_out()

standard_room.book_room()
standard_room.check_in()
standard_room.check_out()
```

Q8: Fitness Club Management System

Define the `Member` class and its subclasses

```
class Member:
    def __init__(self, member_id, name, age, membership_type,
membership_status="Active"):
        self.__member_id = member_id
        self.name = name
        self.age = age
        self.membership_type = membership_type
        self.membership_status = membership_status

    def get_member_id(self):
        return self.__member_id

    def register(self):
        print(f"Member {self.name} registered with membership type
{self.membership_type}.")

    def renew_membership(self):
        if self.membership_status == "Expired":
            self.membership_status = "Active"
            print(f"Membership for {self.name} has been renewed.")
        else:
            print(f"Membership for {self.name} is already active.")

    def cancel_membership(self):
        if self.membership_status == "Active":
            self.membership_status = "Cancelled"
            print(f"Membership for {self.name} has been cancelled.")
        else:
            print(f"Membership for {self.name} is already cancelled
or expired.")

class FamilyMember(Member):
    def __init__(self, member_id, name, age, membership_type,
family_members):
        super().__init__(member_id, name, age, membership_type)
        self.family_members = family_members

class IndividualMember(Member):
    def __init__(self, member_id, name, age, membership_type):
        super().__init__(member_id, name, age, membership_type)

# Testing the Fitness Club Management System
family_member = FamilyMember(1, "John Doe", 40, "Family", ["Jane
Doe", "Junior Doe"])
individual_member = IndividualMember(2, "Jane Smith", 30,
"Individual")

family_member.register()
individual_member.register()

family_member.cancel_membership()
individual_member.renew_membership()
```

Q9: Event Management System

Define the `Event` class and its subclasses

```
class Event:
    def __init__(self, event_id, name, date, time, location):
        self.__event_id = event_id
        self.name = name
        self.date = date
        self.time = time
        self.location = location
        self.__attendees = []

    def get_event_id(self):
        return self.__event_id

    def create_event(self):
        print(f"Event '{self.name}' created for {self.date} at
{self.time} in {self.location}.")

    def add_attendee(self, attendee):
        self.__attendees.append(attendee)
        print(f"Attendee '{attendee}' added to the event
'{self.name}'.")

    def remove_attendee(self, attendee):
        if attendee in self.__attendees:
            self.__attendees.remove(attendee)
            print(f"Attendee '{attendee}' removed from the event
'{self.name}'.")
        else:
            print(f"Attendee '{attendee}' not found in the event
'{self.name}'.")

    def get_total_attendees(self):
        return len(self.__attendees)

class PrivateEvent(Event):
    def __init__(self, event_id, name, date, time, location, host):
        super().__init__(event_id, name, date, time, location)
        self.host = host

class PublicEvent(Event):
    def __init__(self, event_id, name, date, time, location):
        super().__init__(event_id, name, date, time, location)

# Testing the Event Management System
private_event = PrivateEvent(1, "Birthday Party", "2024-07-10",
"18:00", "John's House", "John Doe")
public_event = PublicEvent(2, "Music Concert", "2024-08-20",
"20:00", "City Arena")

private_event.create_event()
public_event.create_event()

private_event.add_attendee("Jane Doe")
private_event.add_attendee("Junior Doe")
print(f"Total attendees: {private_event.get_total_attendees()}")

private_event.remove_attendee("Jane Doe")
print(f"Total attendees: {private_event.get_total_attendees()}")
```

Q10: Airline Reservation System

Define the `Flight` class and its subclasses

```
class Flight:
    def __init__(self, flight_id, flight_number, departure_airport,
arrival_airport, departure_time, arrival_time, available_seats):
        self.__flight_id = flight_id
        self.flight_number = flight_number
        self.departure_airport = departure_airport
        self.arrival_airport = arrival_airport
        self.departure_time = departure_time
        self.arrival_time = arrival_time
        self.__available_seats = available_seats

    def get_flight_id(self):
        return self.__flight_id

    def book_seat(self):
        if self.__available_seats
```