



Republic of the Philippines
SURIGAO DEL NORTE STATE UNIVERSITY
Narciso Street, Surigao City 8400, Philippines



CS - 223 Object-Oriented Programming

MANAGING TYPES OF RESTAURANT

Presented to: Dr. Unife O. Cagas

Prepared by: Shane L. Malasan

BSCS - 2A2

PROJECT DESCRIPTION

This program defines a hierarchy of restaurant classes to control and analyze their sales. The Restaurant class serves as an abstract base magnificence, presenting a blueprint for different sorts of eating places to inherit from. It has attributes for name and each day sales, in conjunction with techniques for showing the name and calculating sales, which must be carried out by subclasses. Two concrete subclasses are described, the FineDiningRestaurant and FastFoodRestaurant.

FineDiningRestaurant extends Restaurant, adding a score attribute. It implements the abstract approach calculate_revenue, considering a tip percentage. FastFoodRestaurant additionally extends Restaurant, with an additional attribute to suggest if it's open 24 hours. Its calculate_revenue approach calculates sales based totally on the assumption of running hours per day. At the cease of this system, instances of each subclasses are created and their techniques are validated. The first-class eating restaurant's name, rating, and revenue over a sure period are displayed, in addition to the short-meals eating place's call, opening hours, and sales over a detailed variety of days.

OBJECTIVES

Modular Design : Development of a modular and scalable program structure using object-oriented programming principles to represent restaurants and their features in a clear and structured way.

Customize and expand: Allow for the creation of restaurants with unique characteristics by defining abstract and concrete units that can be customized and expanded as needed.

Data Integrity and Verification: Implement validation checks with attributes of classes to ensure consistency and reliability of input data, such as minimum name lengths and appropriate rating ranges.

Functionality and Analytics: Provide functionality to display restaurant information, including name, rating (for fine dining restaurants), and opening hours (for fast food restaurants), and calculate revenue based on a variety of factors such as hours of operation, tip percentage and cost plus restaurants.

Demonstration and testing: Demonstrate how the implemented class works by creating instances, configuring attributes, calling methods to demonstrate their behavior and demonstrate that the program successfully meets its objectives.

Code Importance

This code, following the principles of object-oriented programming, is valuable for creating and managing different types of restaurants with ease. By organizing data into classes and methods, it's easy to reuse and maintain. Plus, it ensures that data remains reliable and consistent by validating inputs. With its flexibility, it allows for customization to suit various restaurant needs, making it a handy tool for restaurant owners and managers to analyze revenue and make informed decisions. Overall, it's a great example of how object-oriented programming can simplify complex tasks like restaurant management.

The 4 Principles of OOP:

1. Abstraction: The "Restaurant" class serves as an abstract base class (ABC), defining a blueprint for other restaurant types to inherit from. It declares an abstract method `calculate_revenue()`, which must be implemented by subclasses. This abstraction allows for different types of restaurants to share common behaviors while providing flexibility for customization.

2. Encapsulation: Encapsulation is about bundling data and methods that operate on the data into a single unit, called a class. In this code, data attributes like (image) are encapsulated within their respective classes. Access to these attributes is controlled through property methods, ensuring data integrity and validation.

```
class FineDiningRestaurant(Restaurant):  
    def __init__(self, name, daily_revenue, rating):  
        super().__init__(name, daily_revenue)  
        self._rating = rating
```

3. Inheritance: Both `FineDiningRestaurant` and `FastFoodRestaurant` classes inherit from the `Restaurant` class. They inherit attributes such as `name` and `daily_revenue` and behaviors like `display_name()`.

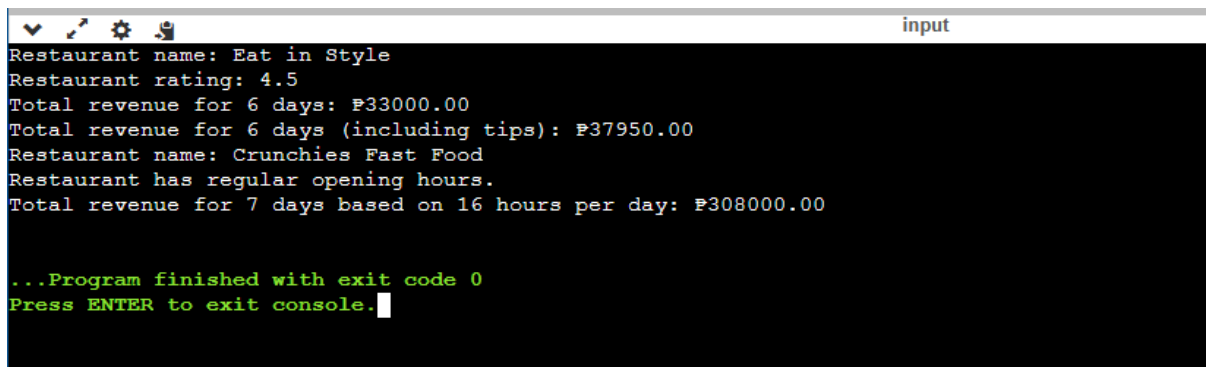
4. Polymorphism: Both `FineDiningRestaurant` and `FastFoodRestaurant` classes override the `calculate_revenue()` method inherited from the `Restaurant` class. Each subclass provides its own implementation of revenue calculation tailored to its specific characteristics. Despite the method having the same name, the behavior varies based on the type of restaurant, showcasing polymorphic behavior.

SOFTWARE USED

This code is written in Python programming language, it showcases Python's object-oriented features, including class definitions, inheritance, and abstract base classes (ABCs), making it a suitable choice for implementing object-oriented programming concepts. I also used GDB as online compiler and debugger.

- The use of standard Python import statements (from abc import ABC, [abstractmethod](#)).
- Classes are defined using Python-specific syntax, such as class ClassName(BaseClassName):.
- The usage of Python-specific decorators such as [@abstractmethod](#) and [@property](#), which signal how to enforce method overrides and manage attribute access, respectively.
- Print statements (print()) and the formatting method ([.2f](#)) are examples of Python's approach to string formatting and console output.

OUTPUT



```
input
Restaurant name: Eat in Style
Restaurant rating: 4.5
Total revenue for 6 days: ₱33000.00
Total revenue for 6 days (including tips): ₱37950.00
Restaurant name: Crunchies Fast Food
Restaurant has regular opening hours.
Total revenue for 7 days based on 16 hours per day: ₱308000.00

...Program finished with exit code 0
Press ENTER to exit console.
```

A **FineDiningRestaurant** object named "Eat in Style" with a daily revenue of **₱5500** and a rating of **4.5** is created.

Calculating and Displaying Revenue for the Fine Dining Restaurant:

`fine_dining_restaurant.calculate_revenue(6):`

This method calculates the revenue over 6 days, including tips at 15%.

Base revenue is calculated as **$5500 * 6 = 33000$**

Tip amount is calculated as **$5500 * 0.15 * 6 = 4950$**

Total revenue, including tips, is **$33000 + 4950 = 37950$**

A **FastFoodRestaurant** object named "**Crunchies Fast Food**" with a daily revenue of **₱2750** and regular **opening hours** (not 24 hours) is created.

Calculating and Displaying Revenue for the Fast Food Restaurant:

fast_food_restaurant.calculate_revenue(7):

This method calculates the revenue over 7 days, assuming regular opening hours (16 hours per day).

Total hours is calculated as **7 * 16 = 112**

Total revenue is calculated as **2750 * 112 = 308000**

The program demonstrates the usage of the `Restaurant` hierarchy by creating instances of `FineDiningRestaurant` and `FastFoodRestaurant`, and calling their respective methods to display information and calculate revenue. The overall purpose of the program is to provide a flexible and extensible way to manage different types of restaurants, allowing for future expansion and the addition of new restaurant types without modifying the existing codebase.

The Program Code

```
from abc import ABC, abstractmethod

class Restaurant(ABC):
    def __init__(self, name, daily_revenue):
        self._name = name
        self._daily_revenue = daily_revenue

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        if len(value) < 3:
            raise ValueError("Name must be at least 3 characters long.")
        self._name = value

    def display_name(self):
        print(f"Restaurant name: {self._name}")

    @abstractmethod
    def calculate_revenue(self, num_days):
        pass # Implementation must be provided by subclasses
```

```
class FineDiningRestaurant(Restaurant):
    def __init__(self, name, daily_revenue, rating):
        super().__init__(name, daily_revenue)
        self._rating = rating

    @property
    def rating(self):
        return self._rating

    @rating.setter
    def rating(self, value):
        if not (1 <= value <= 5):
            raise ValueError("Rating must be between 1 and 5.")
        self._rating = value

    def display_rating(self):
        print(f"Restaurant rating: {self._rating}")

    def calculate_revenue(self, num_days, tip_percentage=0.15):
        base_revenue = self._daily_revenue * num_days
        tip_amount = self._daily_revenue * tip_percentage * num_days
        total_revenue = base_revenue + tip_amount
        print(f"Total revenue for {num_days} days: ₱{base_revenue:.2f}")
        print(f"Total revenue for {num_days} days (including tips): ₱{total_revenue:.2f}")
```

```
class FastFoodRestaurant(Restaurant):
    def __init__(self, name, daily_revenue, is_open_24_hours):
        super().__init__(name, daily_revenue)
        self._is_open_24_hours = is_open_24_hours

    @property
    def is_open_24_hours(self):
        return self._is_open_24_hours

    @is_open_24_hours.setter
    def is_open_24_hours(self, value):
        self._is_open_24_hours = value

    def display_opening_hours(self):
        if self._is_open_24_hours:
            print("Restaurant is open 24 hours.")
        else:
            print("Restaurant has regular opening hours.")

    def calculate_revenue(self, num_days):
        hours_per_day = 24 if self._is_open_24_hours else 16
        total_hours = num_days * hours_per_day
        total_revenue = self._daily_revenue * total_hours
        print(f"Total revenue for {num_days} days based on {hours_per_day} hours per day: ₱{total_revenue:.2f}")
```

```
# Demonstrating the modified classes
fine_dining_restaurant = FineDiningRestaurant("Eat in Style", 5500, 4.5)
fine_dining_restaurant.display_name()
fine_dining_restaurant.display_rating()
fine_dining_restaurant.calculate_revenue(6)

fast_food_restaurant = FastFoodRestaurant("Crunchies Fast Food", 2750, False)
fast_food_restaurant.display_name()
fast_food_restaurant.display_opening_hours()
fast_food_restaurant.calculate_revenue(7)
```

Implementation Details

Abstract Base Class: Restaurant **Purpose:** Serves as a foundation for all types of restaurants. It defines common attributes and methods that all derived classes must implement or inherit. **Attributes:** **_name:** Represents the restaurant's name. It's protected to restrict access to within the class and its subclasses. **_daily_revenue:** Protected attribute storing the daily revenue.

Methods

name property and setter: Ensures the restaurant name is at least 3 characters long, demonstrating basic data validation. **display_name():** Outputs the restaurant's name, illustrating a simple method that derived classes inherit and can use directly. **calculate_revenue(num_days):** An abstract method, forcing subclasses to provide their own implementations. This method is designed to calculate the revenue based on the number of days the restaurant operates, tailored to the specifics of each restaurant type.

Derived Class: FineDiningRestaurant **Inherits:** Restaurant. **Additional Attributes** **_rating:** A specific attribute for fine dining establishments, representing the quality rating between 1 and 5.

Methods

rating property and setter: Implements validation to ensure the rating is between 1 and 5. **display_rating():** Outputs the restaurant's rating. **calculate_revenue(num_days, tip_percentage=0.15):** Overrides the abstract method from Restaurant. It computes total revenue considering base revenue plus tips, reflecting the service charge commonly associated with fine dining.

Derived Class: FastFoodRestaurant **Additional Attributes:** **_is_open_24_hours:** Boolean indicating whether the restaurant operates 24/7.

Methods

is_open_24_hours property and setter: Simple encapsulation to manage the operational hours. **display_opening_hours():** Indicates the operational hours based on the **_is_open_24_hours** flag. **calculate_revenue(num_days):** Overrides the abstract method to calculate revenue based on operating hours (24-hour or regular 16-hour day), showcasing how operational differences affect revenue computation

USER GUIDE

1. Create instances of the restaurants:

```
fine_dining_restaurant = FineDiningRestaurant("Eat in Style", 5500, 4.5)  
fast_food_restaurant = FastFoodRestaurant("Crunchies Fast Food", 2750, False)
```

2. Display restaurant names:

```
fine_dining_restaurant.display_name()  
fast_food_restaurant.display_name()
```

3. Display specific attributes:

```
fine_dining_restaurant.display_rating()  
fast_food_restaurant.display_opening_hours()
```

4. Calculate and display revenue:

```
fine_dining_restaurant.calculate_revenue(6)  
fast_food_restaurant.calculate_revenue(7)
```

Example Output

```
Restaurant name: Eat in Style  
Restaurant rating: 4.5  
Total revenue for 6 days: ₱33000.00  
Total revenue for 6 days (including tips): ₱37950.00  
  
Restaurant name: Crunchies Fast Food  
Restaurant has regular opening hours.  
Total revenue for 7 days based on 16 hours per day: ₱308000.00
```

REFERENCES

[Encapsulation in Python - GeeksforGeeks](#)

[abc — Abstract Base Classes — Python 3.12.3 documentation](#)

[cpython/Lib/abc.py at 3.12 · python/cpython · GitHub](#)