Republic of the Philippines
SURIGAO DEL NORTE STATE UNIVERSITY
Narciso Street, Surigao City 8400, Philippines
"For Nation's Greater"

In Partial Fulfillment of the Requirements for the

CS 223 - Object-Oriented Programming

"FOUR PRINCIPLES OF OBJECT ORIENTED PROGRAMMING"

Presented to:

Dr. Unife O. Cagas
Professor

Prepared by:

Hidalgo, Jessa Mae M.
BSCS 2A2 Student

"Personalized Skincare Product Recommendation Systems in Python"

Project Title

PROJECT DESCRIPTION

My project is a complex Python class hierarchy that aims to accurately simulate the skincare product industry. The SkincareProduct class is fundamentally a foundation that holds important data like name, brand, price, and ingredients. Specialized features like compatibility with different skin types are added to this foundation by subclasses like Moisturizer, Cleanser, and Sunscreen. The display_info() method elegantly highlights the distinctive qualities of each product, providing users with a thorough understanding of its composition and applicability. Your project bridges the gap between theory and practice by providing a flexible framework for clearly and precisely investigating and comprehending skincare items. It does this by instantiating these classes with real-world data and presenting their contents.

OBJECTIVES

1. **Modular Design:** Ensure the project code is structured with a modular approach, facilitating scalability and easy maintenance for future expansions or modifications.

2. **Code Reusability:** Implement inheritance effectively to maximize code reuse, enabling the creation of different skincare product types (e.g., cleansers, moisturizers, sunscreens) while minimizing redundant code.

3. **Encapsulation:** Utilize encapsulation principles to safeguard data integrity by restricting direct access to class attributes, promoting code robustness and reliability.

4. **User-Friendly Display:** Implement a user-friendly display function for each skincare product type, providing comprehensive information including name, brand, price, ingredients, and suitability for specific skin types.

5. **Data Validation:** Incorporate data validation mechanisms to ensure the integrity and correctness of input data (e.g., price validation, ingredient list validation) to prevent errors and enhance code stability.

6. **Documentation and Comments:** Maintain clear and concise documentation and comments throughout the codebase to facilitate understanding, promote collaboration, and streamline future updates or debugging processes.

7. **Testing and Validation:** Conduct thorough testing and validation procedures to ensure the correctness and robustness of the code, covering various use cases and scenarios to identify and rectify potential issues effectively.

IMPORTANCE AND CONTRIBUTION OF THE PROJECT

The provided code presents a simplified model of a Skincare Product Recommendation system, showcasing several implications and contributions across various domains:

- **Consumer Empowerment:** The initiative gives customers essential knowledge about skincare products, such as their ingredients and compatibility for various skin types. This promotes a culture of self-care and wellbeing by enabling customers to make decisions that are well-informed and in line with their skin's needs and preferences.
- **Health and Safety:** By providing detailed ingredient lists and skin type suitability, the project promotes health and safety in skincare routines. Consumers can identify potential allergens or irritants, reducing the risk of adverse reactions and enhancing overall skincare efficacy and safety.
- **Brand Transparency**: The project encourages brand transparency by highlighting product details such as name, brand, and ingredients. This fosters trust between consumers and skincare brands, promoting transparency and accountability in the beauty industry.
- **Educational Resource**: The project serves as an educational resource for individuals interested in skincare. By showcasing the correlation between ingredients and product efficacy, it promotes awareness and understanding of skincare principles, empowering users to make educated decisions about their skincare routines.
- **Customization and Personalization:** With information about skin type suitability, the project facilitates customization and personalization of skincare routines. Users can select products tailored to their specific skin concerns, optimizing skincare outcomes and enhancing user satisfaction.

FOUR PRINCIPLES OF OOP WITH CODE

ENHERITANCE

In my code, you have a base class called SkincareProduct with attributes like name, brand, price, and ingredients. Then, you have subclasses Moisturizer, Cleanser, and Sunscreen that inherit from SkincareProduct.

Each subclass has its own __init__ method to initialize its unique attribute skin_type, in addition to calling the __init__ method of the parent class (SkincareProduct) using super().__init__().

The display_info method is overridden in each subclass to display the information specific to that type of skincare product, along with the skin type it's suitable for. When I create objects like cleanser, moisturizer, and sunscreen, they inherit the attributes and methods from the SkincareProduct class and have their specific attributes initialized through their respective subclass __init__ methods. Finally, I display information about each product using the display_info method, which also takes advantage of method overriding to display specific information for each type of skincare product.

```
class SkincareProduct:
    def __init__(self, name, brand, price, ingredients):
        self.name = name
        self.brand = brand
        self.price = price
        self.ingredients = ingredients

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Brand: {self.brand}")
        print(f"Price: ₱{self.price}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")

class Moisturizer(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Cleanser(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Sunscreen(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

# Creating objects
cleanser = Cleanser("Cleanser", "Cetaphil", 9.99, ["Water", "Cetyl Alcohol", "Propylene Glycol"], "All skin
moisturizer = Moisturizer("Moisturizer", "CeraVe", 15.99, ["Water", "Glycerin", "Hyaluronic Acid"], "Normal
sunscreen = Sunscreen("Sunscreen", "Neutrogena", 12.49, ["Octocrylene", "Avobenzone", "Oxybenzone"], "All s

# Displaying information about the products
print("Cleanser:")
cleanser.display_info()
print("\nMoisturizer:")
moisturizer.display_info()
print("\nSunscreen:")
```

ENCAPSULATION

Encapsulation in my code is evident through the use of classes to bundle together both data and methods that operate on that data. Each class, such as SkincareProduct, Moisturizer, Cleanser, and Sunscreen, encapsulates its own set of attributes like name, brand, price, ingredients, and skin_type. Methods within each class, like display_info, encapsulate functionality specific to each type of skincare product, providing a clean interface for interacting with and manipulating the data. While attributes are currently public, encapsulation could be further enhanced by making them private and providing getter and setter methods to control access and modification. This encapsulated design promotes data hiding, abstraction, and modularity, enhancing the organization and maintainability of your code.

```python
class SkincareProduct:
    def __init__(self, name, brand, price, ingredients):
        self.name = name
        self.brand = brand
        self.price = price
        self.ingredients = ingredients

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Brand: {self.brand}")
        print(f"Price: ₱{self.price}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")

class Moisturizer(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Cleanser(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Sunscreen(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

# Creating objects
cleanser = Cleanser("Cleanser", "Cetaphil", 9.99, ["Water", "Cetyl Alcohol", "Propylene Glycol"], "All skin
moisturizer = Moisturizer("Moisturizer", "CeraVe", 15.99, ["Water", "Glycerin", "Hyaluronic Acid"], "Normal
sunscreen = Sunscreen("Sunscreen", "Neutrogena", 12.49, ["Octocrylene", "Avobenzone", "Oxybenzone"], "All s

# Displaying information about the products
print("Cleanser:")
cleanser.display_info()
print("\nMoisturizer:")
moisturizer.display_info()
print("\nSunscreen:")
sunscreen.display_info()
```

POLYMORPHISM

Each subclass (Moisturizer, Cleanser, Sunscreen) overrides the display_info method of the base class (SkincareProduct). Despite having the same method name, each subclass provides its own implementation of display_info, allowing for different behavior depending on the type of skincare

product. This means that when you call display_info on an object of any subclass, the appropriate version of the method is executed based on the type of object, showcasing polymorphic behavior. For example, the display_info method in the Cleanser subclass prints information about a cleanser product along with its suitable skin type, while the same method in the Sunscreen subclass prints information specific to sunscreen products. This flexibility in method implementation based on the object's type is a key aspect of polymorphism.

```python
class SkincareProduct:
    def __init__(self, name, brand, price, ingredients):
        self.name = name
        self.brand = brand
        self.price = price
        self.ingredients = ingredients

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Brand: {self.brand}")
        print(f"Price: ₱{self.price}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")

class Moisturizer(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Cleanser(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Sunscreen(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

# Creating objects
cleanser = Cleanser("Cleanser", "Cetaphil", 9.99, ["Water", "Cetyl Alcohol", "Propylene Glycol"], "All skin
moisturizer = Moisturizer("Moisturizer", "CeraVe", 15.99, ["Water", "Glycerin", "Hyaluronic Acid"], "Normal
sunscreen = Sunscreen("Sunscreen", "Neutrogena", 12.49, ["Octocrylene", "Avobenzone", "Oxybenzone"], "All s

# Displaying information about the products
print("Cleanser:")
cleanser.display_info()
print("\nMoisturizer:")
moisturizer.display_info()
print("\nSunscreen:")
```

ABSTRACTION

Abstraction, as demonstrated in my code, involves concealing complex implementation details and presenting only the essential features of objects. Through the class structure and method definitions,

your code achieves abstraction effectively. The SkincareProduct class encapsulates common attributes like name, brand, price, and ingredients, abstracting away the complexity of how these details are stored or managed internally. The display_info method further abstracts the presentation of product information, providing a high-level interface for displaying details about a skincare product without exposing the underlying data structure. Subclasses (Moisturizer, Cleanser, Sunscreen) build upon this abstraction by adding specific details relevant to each type of product, such as the suitable skin type. This abstraction promotes clarity and simplifies the representation of skincare products, enhancing the readability and maintainability of my codebase.

```
class SkincareProduct:
    def __init__(self, name, brand, price, ingredients):
        self.name = name
        self.brand = brand
        self.price = price
        self.ingredients = ingredients

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Brand: {self.brand}")
        print(f"Price: ₱{self.price}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")

class Moisturizer(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Cleanser(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Sunscreen(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

# Creating objects
cleanser = Cleanser("Cleanser", "Cetaphil", 9.99, ["Water", "Cetyl Alcohol", "Propylene Glycol"], "All skin
moisturizer = Moisturizer("Moisturizer", "CeraVe", 15.99, ["Water", "Glycerin", "Hyaluronic Acid"], "Normal
sunscreen = Sunscreen("Sunscreen", "Neutrogena", 12.49, ["Octocrylene", "Avobenzone", "Oxybenzone"], "All s

# Displaying information about the products
print("Cleanser:")
cleanser.display_info()
print("\nMoisturizer:")
moisturizer.display_info()
print("\nSunscreen:")
sunscreen.display_info()
```

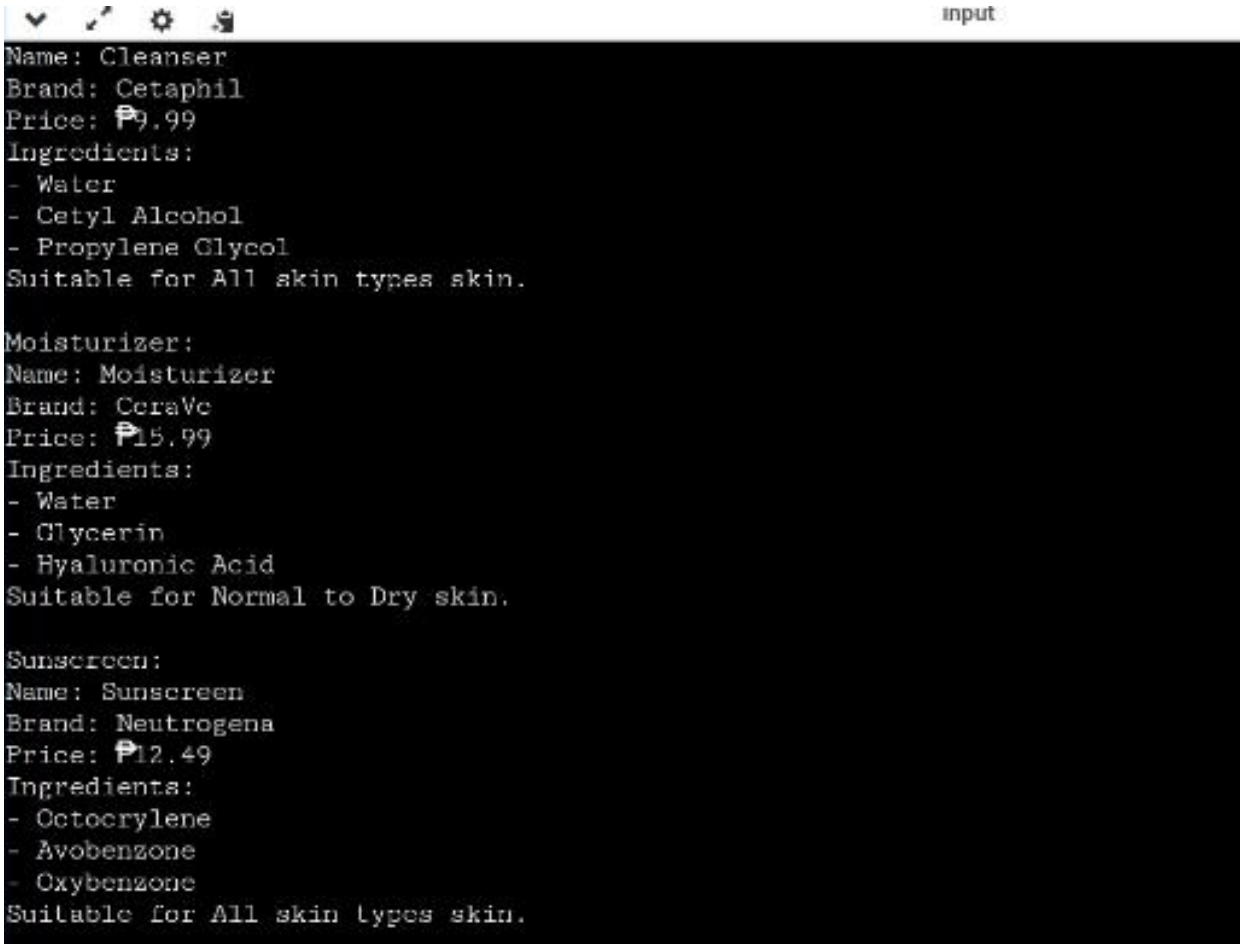
SOFTWARE USED:

- ONLINE GDB

HARDWARE USED:

- MOBILE PHONE

OUTPUT



The output of my code presents detailed information about three skincare products: a cleanser, a moisturizer, and a sunscreen. Each product's name, brand, price, and list of ingredients are displayed, followed by the type of skin it is suitable for. This output demonstrates the polymorphic behavior of the display_info method, as each subclass (Cleanser, Moisturizer, Sunscreen) overrides the method inherited from the base class (SkincareProduct) to provide customized information specific to its type of product. By invoking the display_info method on each object, the system dynamically selects the appropriate implementation based on the type of product, showcasing the flexibility and extensibility of object-oriented programming principles.

CODE

```python
class SkincareProduct:
    def __init__(self, name, brand, price, ingredients):
        self.name = name
        self.brand = brand
        self.price = price
        self.ingredients = ingredients

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Brand: {self.brand}")
        print(f"Price: ₱{self.price}")
        print("Ingredients:")
        for ingredient in self.ingredients:
            print(f"- {ingredient}")

class Moisturizer(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Cleanser(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

class Sunscreen(SkincareProduct):
    def __init__(self, name, brand, price, ingredients, skin_type):
        super().__init__(name, brand, price, ingredients)
        self.skin_type = skin_type

    def display_info(self):
        super().display_info()
        print(f"Suitable for {self.skin_type} skin.")

# Creating objects
cleanser = Cleanser("Cleanser", "Cetaphil", 9.99, ["Water", "Cetyl Alcohol", "Propylene Glycol"], "All skin
moisturizer = Moisturizer("Moisturizer", "CeraVe", 15.99, ["Water", "Glycerin", "Hyaluronic Acid"], "Normal
sunscreen = Sunscreen("Sunscreen", "Neutrogena", 12.49, ["Octocrylene", "Avobenzone", "Oxybenzone"], "All s

# Displaying information about the products
print("Cleanser:")
cleanser.display_info()
print("\nMoisturizer:")
moisturizer.display_info()
print("\nSunscreen:")
sunscreen.display_info()
```

USER GUIDE: **Skincare Product Recommendation Systems**

Welcome to the Skincare Product Management System! This system allows you to manage and display information about various skincare products, including moisturizers, cleansers, and sunscreens.

Getting Started:

1. Navigate to the OnlineGDB website: OnlineGDB

2. Create a new Python project.

3. Copy and paste the provided code into the code editor.

Using the System:

1. Define Skincare Products**:**

- Create instances of skincare products by providing the necessary details such as name, brand, price, ingredients, and suitable skin type.
- Use the provided classes Moisturizer, Cleanser, and Sunscreen to create objects representing different types of skincare products.

2. Display Product Information:

- Call the display_info() method on each skincare product object to view its details.
- The system will print out information such as name, brand, price, ingredients, and suitability for specific skin types.

3. Customize and Extend:

- Customize the system by adding new types of skincare products or modifying existing ones.
- Extend functionality by adding new methods or attributes to the existing classes.

REFERENCES

- https://www.onlinegdb.com/

- https://python.org/
- https://chatgpt.com/
- https://www.blackbox.ai/