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 V

Presented by:

Ma. Juvywin L. Buenavista
BSCS 2A2 Student

" Bag Types"

Project Title

Project Description

This code demonstrates the concept of abstraction and inheritance in Python. It defines an abstract class called "Bag" which enforces a common interface for all types of bags. The abstract class has one abstract method called "display_info". The code then defines two classes, "Handbag" and "ToteBag", both of which inherit from the "Bag" class. The "Handbag" class has encapsulated attributes such as brand, material, color, and size. It also has getter and setter methods for the encapsulated attributes. The "Handbag" class overrides the "display_info" method from the abstract class to display information about the handbag. Similarly, the "ToteBag" class has encapsulated attributes such as brand, material, color, size, and strap length. It overrides the "display_info" method to display information about the tote bag. The code then creates instances of the "Handbag" and "ToteBag" classes, and displays information about each bag using the "display_info" method.

Objectives:

1. Demonstrate the concept of abstraction by creating an abstract class called "Bag" with an abstract method called "display_info". This enforces a common interface for all types of bags.

2. Implement inheritance by creating two classes, "Handbag" and "ToteBag", that inherit from the "Bag" class. This allows the classes to inherit the common interface and behavior defined in the abstract class.

3. Encapsulate attributes within the classes to ensure data privacy and provide access to them through getter and setter methods. This promotes encapsulation and data abstraction.

4. Override the "display_info" method in the derived classes to display specific information about each type of bag. This demonstrates polymorphism, as the method is implemented differently in each class but can be called using the same interface.

5. Create instances of the "Handbag" and "ToteBag" classes and display information about each bag using the "display_info" method. This showcases the usage of the abstract class, inheritance, and polymorphism to create and manipulate objects.

Importance and Contribution of the Project

The project showcases a clear understanding of object-oriented programming principles, specifically inheritance, encapsulation, and abstract classes. It defines an abstract base class Bag and two derived classes Handbag and ToteBag. This hierarchy enforces a common interface for all bag types, allowing for a more organized code structure and easier maintenance.

Four Principles of Object-Oriented Programming with code

Class:

A class is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have. In the given code, the classes "Bag", "Handbag", and "ToteBag" are defined.

```
class Bag(ABC):

class Handbag(Bag):

class ToteBag(Bag):
```

Objects:

Objects are instances of a class. They are created based on the blueprint provided by the class. In the code, instances of the classes "Handbag" and "ToteBag" are created using the "handbag1", "handbag2", "totebag1", and "totebag2" variables.

```
handbag1 = Handbag("Louis Vuitton", "leather", "black", "medium")
handbag2 = Handbag("Gucci", "suede", "brown", "small")
totebag1 = ToteBag("Prada", "canvas", "blue", "large", 24)
totebag2 = ToteBag("Michael Kors", "nylon", "red", "extra large", 28)
```

Inheritance:

Inheritance is a mechanism that allows a class to inherit properties and behaviors from another class. The derived class (subclass) inherits the attributes and methods of the base class (superclass). In the code, both the "Handbag" and "ToteBag" classes inherit from the "Bag" class, which defines the common interface and behavior for all types of bags.

```
class Handbag(Bag):
class ToteBag(Bag):
```

Encapsulation:

Encapsulation is the process of hiding the internal details of an object and providing access to them through methods. It helps in achieving data abstraction and data security. In the code, the attributes of the "Handbag" and "ToteBag" classes (such as brand, material, color, etc.) are encapsulated by using private variables and providing getter and setter methods to access and modify them.

```
class Handbag(Bag):
    def __init__(self, brand, material, color, size):
        self.__brand = brand
        self.__material = material
        self.__color = color
        self.__size = size

    def get_brand(self):
        return self.__brand

    def set_brand(self, brand):
        self.__brand = brand
```

Polymorphism:

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It allows different classes to define their own implementation of methods with the same name, providing different behaviors based on the type of object. In the code, both the "Handbag" and "ToteBag" classes override the "display_info" method from the "Bag" class to display specific information about each type of bag.

```
class Handbag(Bag):
    def display_info(self):
        return f"A {self.__color} {self.__size}-sized {self.__material} handbag by {self.__brand}"

class ToteBag(Bag):
    def display_info(self):
        return f"A {self.__color} {self.__size}-sized
        {self.__material} tote bag by {self.__brand} with strap length {self.__strap_length} inches"
```

Abstraction:

Abstraction is the process of hiding unnecessary details and exposing only essential features. It allows us to focus on the functionality of an object rather than its implementation. In the code, the "Bag" class is an abstract class that defines the common interface for all bags, ensuring that all derived classes implement the "display_info" method.

```
from abc import ABC

class Bag(ABC):
    def display_info(self):
        pass
```

Hardware and Software Used

Hardware:

- Laptop
- Cellphone

Software:

- Visual Studio Code
- Online GDB

Output:

```
PS C:\Users\Admin\Documents\Buenavista> & C:/ProgramData/anaconda3/python.exe c:/Users/Admin/Documents/Buenavista/OOP
A black medium-sized leather handbag by Louis Vuitton
--------------------------------------------------
A brown small-sized suede handbag by Gucci
--------------------------------------------------
A blue large-sized canvas tote bag by Prada with strap length 24 inches
--------------------------------------------------
A red extra large-sized nylon tote bag by Michael Kors with strap length 28 inches
```

Description:

- handbag1.display_info(): Outputs information about handbag1, which is a medium-sized leather handbag by Louis Vuitton, with a black color.
- handbag2.display_info(): Outputs information about handbag2, which is a small-sized suede handbag by Gucci, with a brown color.
- totebag1.display_info(): Outputs information about totebag1, which is a large-sized canvas tote bag by Prada, with a blue color and a strap length of 24 inches.
- totebag2.display_info(): Outputs information about totebag2, which is an extra large-sized nylon tote bag by Michael Kors, with a red color and a strap length of 28 inches.

Each output provides detailed information about the respective bag, including brand, material, color, size, and strap length (for tote bags). This demonstrates the polymorphic behavior of the display_info() method, as it generates different output depending on the type of bag object it is called on.

Code Documentation:

```
from abc import ABC # Import the Abstract Base Class and abstractmethod to create abstract classes and methods


# Define an abstract class Bag to enforce a common interface
class Bag(ABC):  # Abstract base class for different types of bags
    def display_info(self):  # Abstract method to display information about the bag
```

```
    pass  # No implementation in the base class, as it is meant to be overridden by subclasses


# Define the Handbag class inheriting from Bag
class Handbag(Bag):  # Subclass of Bag, representing a handbag
    def __init__(self, brand, material, color, size):  # Constructor to initialize a handbag
        # Encapsulated attributes to restrict direct access
        self.__brand = brand
        self.__material = material
        self.__color = color
        self.__size = size

    def get_brand(self):  # Getter method to access the brand attribute
        return self.__brand


    def set_brand(self, brand):  # Setter method to change the brand attribute
        self.__brand = brand

    def display_info(self):  # Implementation of the abstract method from Bag class

        return f"A {self.__color} {self.__size}-sized {self.__material} handbag by {self.__brand}"  # Return a formatted string describing the handbag

# Define a derived class ToteBag inheriting from Bag
class ToteBag(Bag):  # Subclass of Bag, representing a tote bag
    # Constructor with additional attribute for strap length
    def __init__(self, brand, material, color, size, strap_length):
        # Encapsulated attributes
        self.__brand = brand
```

```
        self.__material = material
        self.__color = color
        self.__size = size
        self.__strap_length = strap_length

    def display_info(self):  # Implementation of the abstract method from Bag class
        return f"A {self.__color} {self.__size}-sized {self.__material} tote bag by {self.__brand} with strap length {self.__strap_length} inches"

# Create instances of Handbag and ToteBag
handbag1 = Handbag("Louis Vuitton", "leather", "black", "medium")
handbag2 = Handbag("Gucci", "suede", "brown", "small")
totebag1 = ToteBag("Prada", "canvas", "blue", "large", 24)
totebag2 = ToteBag("Michael Kors", "nylon", "red", "extra large", 28)
# Display information about each bag
print(handbag1.display_info())
print("-----------------------------------------------------")
print(handbag2.display_info())
print("-----------------------------------------------------")
print(totebag1.display_info())
print("-----------------------------------------------------")
print(totebag2.display_info())
```

User Guide:

Step 1: Importing the Required Module

Ensure that you import the necessary module to use abstract classes and methods.

Step 2: Understanding the Code Structure

The provided code defines two classes: Handbag and ToteBag, both inheriting from the abstract base class Bag. Each class represents a specific type of bag and provides information about the bag.

Step 3: Creating Instances of Bag Classes

Instantiate objects of the Handbag and ToteBag classes by providing the required attributes such as brand, material, color, size, and strap length (for ToteBag).

Step 4: Displaying Bag Information

Call the appropriate method on each bag object to print information about each bag.

References:

https://chatgpt.com/c/03e18662-9eb8-4b95-a6f0-1cddf4f6bd5e

https://www.w3schools.com/python/default.asp