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

Korean Actors

Presented to:

Dr. Unife O. Cagas
Professor

Prepared by:

Diaz, Katrina Carla M.
Student

BSCS-2A: Computer Science

May. 2024

Project Title:

Korean Actors

Project Description:

This Python code defines two classes: `kActor` for general actors and `kLeadActor` for lead actors. Actors are initialized with attributes like name, age, and drama. Lead actors additionally have a lead role attribute. The code demonstrates inheretance, Encapsulation, Abstraction, and Polymorphism. It creates instances of these classes and showcases their introductions, including their attributes.

Objectives:

- To analyze survey data collected from students regarding their preferences in mobile devices.
- To identify popular brands, models, and hardware specifications among students.
- To generate reports and visualizations that provide actionable insights for mobile marketing strategies.
- To create a user-friendly interface for uploading survey data and viewing analysis results.

Importance and Contribution:

This project provides a useful illustration of the essential ideas of object-oriented programming (OOP). It provides a good illustration of the significance of ideas like inheritance, encapsulation, and method overriding in software development. It improves code reuse and data management by classifying actor-related data into structured groups. Furthermore, subclassing enables specificity and customisation, accommodating various actor roles while preserving a common interface. Overall, this code effectively aids in the practical application of OOP ideas being learned.

Hardware & Software Used:

Hardware:

- Computer or Cellphone

Software:

- GDB

Four Principles of Object Oriented Programming

Inheritance:

```
class kLeadActor(kActor):
```

- Shows that kLeadActor inherits attributes and methods from the kActor class. This means kLeadActor can access and use all functionalities of kActor while also having its own unique features.

Polymorphism:

```
def introduce(self):
```

- Let's multiple classes have methods with the same name but different implementations, which illustrates polymorphism. This makes it possible for those classes' objects to be handled consistently using a single interface.

Encapsulation:

```
def __init__(self, name, age, drama):
def __init__(self, name, age, drama, lead_role):
```

- Represent encapsulation by initializing and bundling object attributes within their respective classes. This ensures that the internal state of the objects is hidden and can only be accessed or modified through defined methods, promoting data integrity and abstraction.

Abstraction:

```
def introduce(self):
```

- Represents abstraction by providing a simplified interface to retrieve actor information, hiding the internal implementation details of how the introduction is done. It allows users to interact with objects without needing to understand the underlying complexities.

Code Documentation:

```
class kActor:
    def __init__(self, name, age, drama):  # Encapsulation: Constructor to initialize object state
        self.name = name
        self.age = age
        self.drama = drama

    def introduce(self):  # Abstraction: Exposes only necessary features to interact with the class
        print(f"Name: {self.name}")
        print(f"Age: {self.age} years old")
        print(f"Drama: {self.drama}")

class kLeadActor(kActor):  # Inheritance: Build new classes on existing ones
    def __init__(self, name, age, drama, lead_role):  # Encapsulation: Inherits and extends encapsulated behavior
        super().__init__(name, age, drama)
        self.lead_role = lead_role

    def introduce(self):  # Polymorphism: Method overriding to customize behavior in subclasses
        super().introduce()
        print(f"Lead Role: {self.lead_role}")

# Create kActor objects
actor1 = kActor("Lee Jae-Wook", 25, "Alchemy of Souls")  # Classes and Objects: Creating an object of kActor
actor2 = kActor("Cha Eun-Woo", 27, "Wonderful World")    # Classes and Objects: Creating an object of kActor
actor3 = kActor("Byeon Woo-seok", 32, "Lovely Runner")   # Classes and Objects: Creating an object of kActor

# Create kLeadActor object
lead_actor1 = kLeadActor("Park Seo-Joon", 33, "Itaewon Class", "Park Sae-Ro-Yi")
# Classes and Objects: Creating an object of kLeadActor

# Access object attributes
print("==========================")
lead_actor1.introduce()
print("==========================")
actor1.introduce()
print("==========================")
actor2.introduce()
print("==========================")
actor3.introduce()
print("==========================")
```

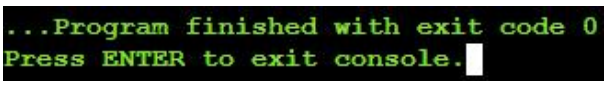
User Guide:

Step 1:

- Start the program by pressing the Run button.

Step 2:

- You'll see the Program Finished.



Output:

```
============================
Name: Park Seo-Joon
Age: 33 years old
Drama: Itaewon Class
Lead Role: Park Sae-Ro-Yi
============================
Name: Lee Jae-Wook
Age: 25 years old
Drama: Alchemy of Souls
============================
Name: Cha Eun-Woo
Age: 27 years old
Drama: Wonderful World
============================
Name: Byeon Woo-seok
Age: 32 years old
Drama: Lovely Runner
============================
```

Description:

The code defines classes for actors (kActor) and lead actors (kLeadActor). It showcases OOP concepts like inheritance, encapsulation, polymorphism, and abstraction. Objects are created to represent actors, their details are printed, demonstrating the organization and management of actor information.

Conclusion:

In conclusion, this code effectively demonstrates key Object-Oriented Programming (OOP) principles such as inheritance, encapsulation, polymorphism, and abstraction. By defining classes for actors and lead actors, it showcases how to organize and manage data in a structured and modular way. Through the creation of objects and the utilization of class methods, the code highlights how OOP can provide a clear and concise means of representing real-world entities and their behaviors. Overall, this example serves as a valuable educational resource for understanding and applying OOP concepts in Python programming.

References:

GDB Code link: https://onlinegdb.com/3UFcq2Rk-

Encapsulation: https://docs.python.org/3/tutorial/classes.html (Section 9.3)

Inheritance: https://docs.python.org/3/tutorial/classes.html (Section 9.2)

Polymorphism: https://stackoverflow.com/questions/12031018/overriding-in-python

super(): https://realpython.com/python-super/#an-overview-of-pythons-super-function