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:

Josephine D. Dingding
BSCS 2A2 Student

" University Structure with Student, Faculty, and Staff"

Project Title

Project Description

Universities are complex environments with various roles contributing to their functioning. This code snippet models a simplified university structure, incorporating three key groups of people: students, faculty, and staff. Each group has unique attributes and behaviors, but they share commonalities that make them part of the broader university community. The code uses object-oriented programming concepts to define these roles and their characteristics. At the core, there's an abstract base class, Person, which provides common attributes and behaviors shared by all people in the university. This class acts as a blueprint for derived classes, allowing for both inheritance and polymorphism.

Objectives:

1. Define an abstract base class that serves as a common structure for different roles in a university, including shared attributes like name and age. This class should facilitate inheritance and be abstract enough to allow flexibility in derived classes.

2. Extend the base class to create specific subclasses for students, faculty, and staff. Each subclass should have unique attributes and behavior relevant to its role, such as student_id for students, department for faculty, and role for staff.

3. Use polymorphism to allow subclasses to define specific behavior while adhering to a common interface. The describe method in each subclass should return a unique description that reflects the role's characteristics.

4. Create instances of Student, Faculty, and Staff to demonstrate the structure's functionality. Use these instances to show how the describe method can vary among subclasses while retaining a common approach to information retrieval.

5. Write code that is clear, well-documented, and easy to understand. Use consistent naming conventions and avoid code redundancy. This objective helps ensure the code is maintainable and can be easily extended or modified in the future.

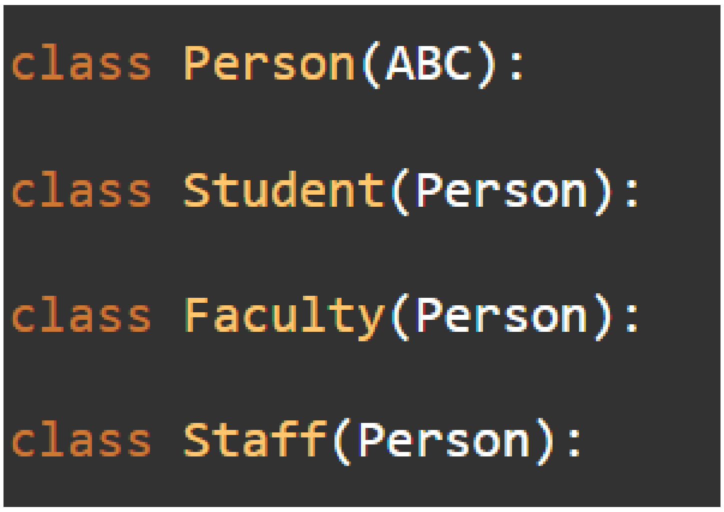
Importance and Contribution of the Project

This project is an example of object-oriented programming in Python, demonstrating the use of abstract base classes, inheritance, and encapsulation. The project defines a base class Person and three derived classes Student, Faculty, and Staff. Each class has its own attributes and methods, and the derived classes inherit the attributes and methods of the base class. The importance of this project lies in its demonstration of the fundamental concepts of object-oriented programming, which are widely used in software development. By creating an abstract base class Person, the project enforces a common interface for all derived classes, ensuring that they have a consistent structure and behavior. The use of encapsulation, through the use of protected attributes, ensures that the internal state of the objects is hidden from the outside world, making the code more robust and maintainable.

Four Principles of Object-Oriented Programming with code

Class:

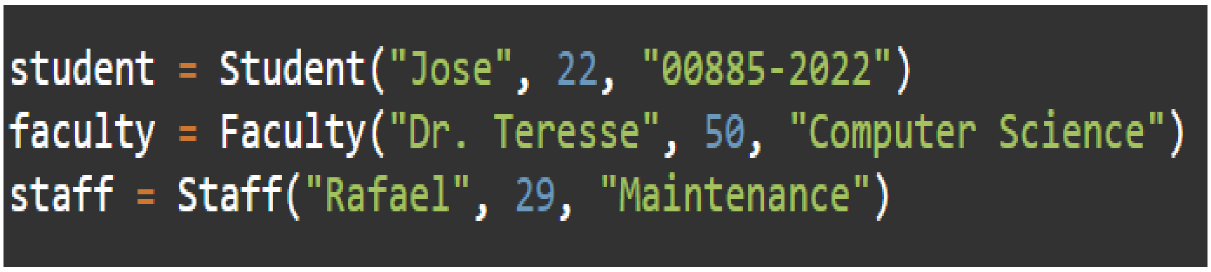
You have defined classes for each type of individual within the university: Student, Faculty, and Staff. These classes represent the blueprints for creating objects.



```python
class Person(ABC):

class Student(Person):

class Faculty(Person):

class Staff(Person):
```

Object:

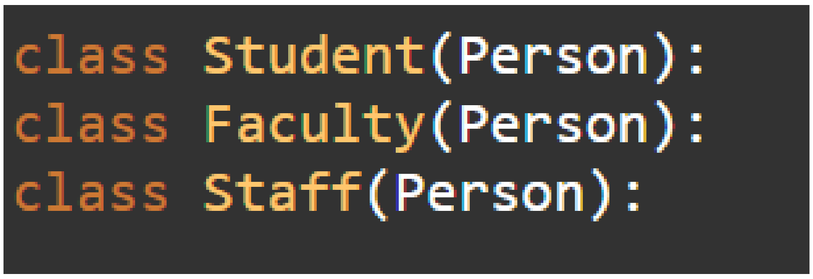
You create objects based on these classes, such as student, faculty, and staff, which represent specific instances of students, faculty members, and staff members, respectively.



```python
student = Student("Jose", 22, "00885-2022")
faculty = Faculty("Dr. Teresse", 50, "Computer Science")
staff = Staff("Rafael", 29, "Maintenance")
```

Inheritance:

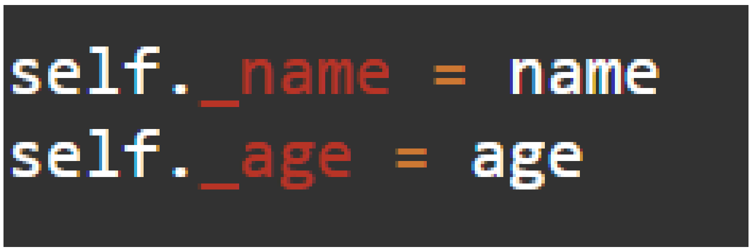
The Student, Faculty, and Staff classes inherit from the Person class. This allows them to inherit common attributes and methods from the Person class, reducing code duplication.



```python
class Student(Person):
class Faculty(Person):
class Staff(Person):
```

Encapsulation:

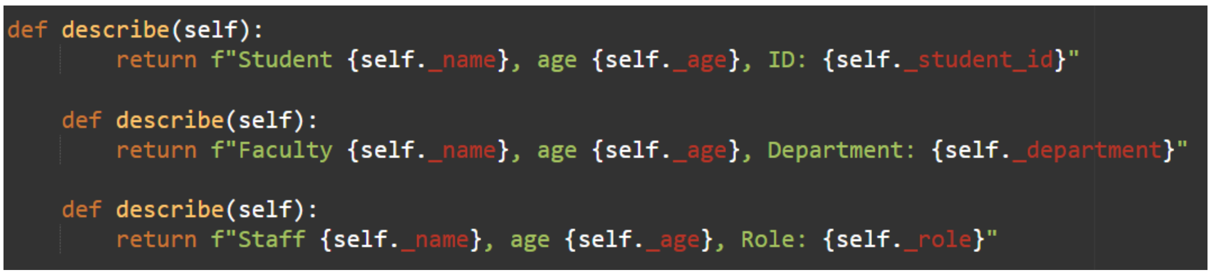
You've used encapsulation by defining attributes (e.g., _name, _age, _student_id, _department, _role) as protected (by prefixing them with an underscore). This means they can't be accessed directly from outside the class, promoting data integrity and hiding implementation details.



```python
self._name = name
self._age = age
```

Polymorphism:

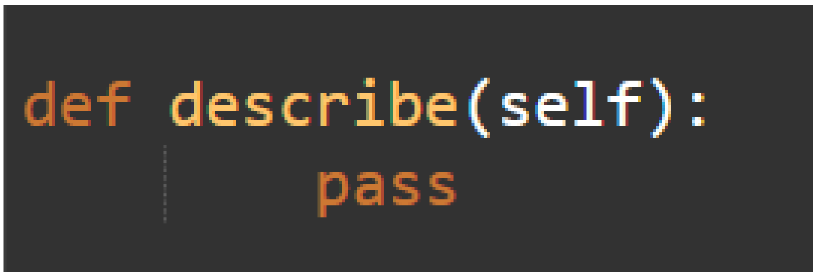
Polymorphism is demonstrated through method overriding. Each subclass (Student, Faculty, Staff) provides its own implementation of the describe() method, allowing for different behaviors for each type of person while using a common interface.



```python
def describe(self):
        return f"Student {self._name}, age {self._age}, ID: {self._student_id}"

    def describe(self):
        return f"Faculty {self._name}, age {self._age}, Department: {self._department}"

    def describe(self):
        return f"Staff {self._name}, age {self._age}, Role: {self._role}"
```

Abstraction:

You've abstracted common behaviors and attributes into the Person class. This class defines common methods like get_name(), set_name(), get_age(), and describe() which are inherited by subclasses. This abstraction allows you to work with generic Person objects without needing to know the specific details of each subclass.



```python
def describe(self):
        pass
```

Hardware and Software Used

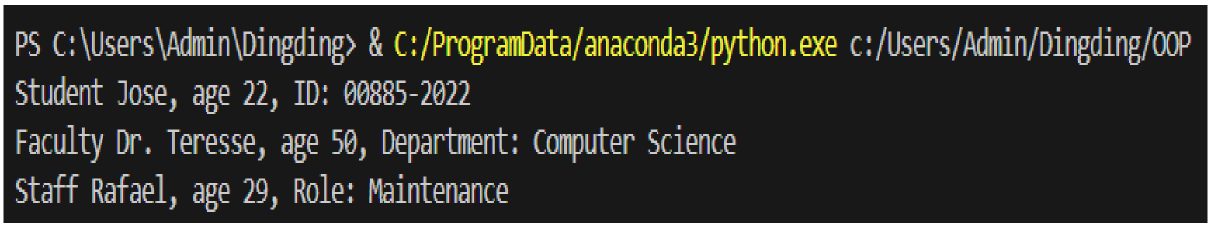
Hardware:

- Laptop
- Cellphone

Software:

- Visual Studio Code
- Online GDB

Output:



```
PS C:\Users\Admin\Dingding> & C:/ProgramData/anaconda3/python.exe c:/Users/Admin/Dingding/OOP
Student Jose, age 22, ID: 00885-2022
Faculty Dr. Teresse, age 50, Department: Computer Science
Staff Rafael, age 29, Role: Maintenance
```

Description:

The Person class is an abstract base class (although it doesn't enforce any abstract methods) that serves as the base for Student, Faculty, and Staff classes. Each subclass has specific attributes (student_id for Student, department for Faculty, and role for Staff) along with a method to describe the person. Instances of Student, Faculty, and Staff are created with their respective attributes, and these instances are added to the list people. The loop iterates over the people list, calling the describe method on each instance, which outputs information specific to each class.

Code Documentation:

```
# Import the Abstract Base Class module, which allows us to define abstract base classes.

from abc import ABC

# Define an abstract base class `Person`.

class Person(ABC):

    # Constructor for the `Person` class, which initializes `name` and `age`.

    def __init__(self, name, age):

        # Store the `name` in a protected attribute.

        self._name = name

        # Store the `age` in a protected attribute.

        self._age = age


    # Method to get the `name`.

    def get_name(self):

        return self._name
```

```
# Method to set the `name`.
    def set_name(self, name):
        self._name = name

    # Method to get the `age`.
    def get_age(self):
        return self._age

    # Abstract method to be implemented by subclasses for providing a description.
    def describe(self):
        pass

# Define a `Student` class that inherits from `Person`.
class Student(Person):
# Constructor for the `Student` class, which initializes `name`, `age`, and
`student_id`.
    def __init__(self, name, age, student_id):
        # Call the constructor of the base class `Person`.
        super().__init__(name, age)
        # Store the `student_id` in a protected attribute.
        self._student_id = student_id

    # Method to get the `student_id`.
    def get_student_id(self):
        return self._student_id

    # Method to set the `student_id`.
    def set_student_id(self, student_id):
        self._student_id = student_id
```

```
 # Method to provide a description of the `Student`.
    def describe(self):
        return f"Student {self._name}, age {self._age}, ID: {self._student_id}"


# Define a `Faculty` class that inherits from `Person`.
class Faculty(Person):
    # Constructor for the `Faculty` class, which initializes `name`, `age`, and
`department`.
    def __init__(name, age, department):
        # Call the constructor of the base class `Person`.
super().__init__(name, age)


 # Store the `department` in a protected attribute.
        self._department = department


    # Method to get the `department`.
    def get_department(self):
        return self._department


    # Method to set the `department`.
    def set_department(self, department):
        self._department = department


    # Method to provide a description of the `Faculty`.
    def describe(self):
        return f"Faculty {self._name}, age {self._age}, Department: {self._department}"


# Define a `Staff` class that inherits from `Person`.
class Staff(Person):
    # Constructor for the `Staff` class, which initializes `name`, `age`, and `role`.
 def __init__(name, age, role):
        # Call the constructor of the base class `Person`.
        super().__init__(name, age)
        # Store the `role` in a protected attribute.
        self._role = role

    # Method to get the `role`.
    def get_role(self):
        return self._role

 # Method to set the `role`.
    def set_role(self, role):
        self._role = role

    # Method to provide a description of the `Staff`.
    def describe(self):
        return f"Staff {self._name}, age {self._age}, Role: {self._role}"

# Create instances of `Student`, `Faculty`, and `Staff`.
student = Student("Jose", 22, "00885-2022")
faculty = Faculty("Dr. Teresse", 50, "Computer Science")
staff = Staff("Rafael", 29, "Maintenance")

# Create a list to hold the instances of `Student`, `Faculty`, and `Staff`.
people = [student, faculty, staff]
# Iterate over the list `people`, calling the `describe` method on each element.
for person in people:
# Print the description of each person.
    print(person.describe())
```

User Guide:

- This code allows you to model different types of individuals (students, faculty, and staff) in an educational setting.

To use this code:

- Create instances of Student, Faculty, and Staff, providing the required parameters (name, age, and specific attributes like student_id, department, or role).
- Add these instances to a list.
- Iterate over the list and call the describe() method on each instance to get a description of each person.

References:

W3Schools. "Python Tutorial". Available online: https://www.w3schools.com/python/default.asp

ChatGPT. "OpenAI ChatGPT Demo". Available online: https://chatgpt.com/?oai-dm=1