Object-Oriented Programming: Person Class Hierarchy

1. Aim

The aim of this project is to implement a robust and modular class hierarchy using the principles of Object-Oriented Programming (OOP), specifically focusing on **inheritance**, by creating a base class (Person) and two derived classes (Student and Teacher).

2. Objective

Upon successful completion, the project will achieve the following objectives:

1. Define a superclass (Person) to encapsulate common attributes and methods shared by all individuals.
2. Create specialized subclasses (Student and Teacher) that inherit all properties from the Person class, thereby demonstrating **code reusability**.
3. Implement unique attributes and methods in the subclasses to show **specialization**.
4. Confirm the correct calling of parent class constructors using the super() function.

3. Theory and Core OOP Concepts

A. Inheritance

Inheritance is a mechanism that allows a new class (subclass or derived class) to inherit properties (attributes) and behavior (methods) from an existing class (superclass or base class). This creates an "is-a" relationship, e.g., a Student *is a* Person.

B. Class Structure

- **Superclass: Person**
  - **Attributes:** name, age.
  - **Methods:** __init__, introduce.
- **Subclass 1: Student**
  - **Inherits from:** Person.
  - **Additional Attributes:** student_id, major.
  - **Methods:** Overrides __init__, adds study_status.
- **Subclass 2: Teacher**
  - **Inherits from:** Person.
  - **Additional Attributes:** employee_id, subject.
  - **Methods:** Overrides __init__, adds teaching_load.

C. The super() Function

The super() function is used in the subclass constructor (__init__) to call the constructor of the parent class. This ensures that the parent class's attributes (like name and age) are initialized correctly before the subclass initializes its own specific attributes.

4. Procedure (Python Implementation)

1. **Define the Base Class:** Create the Person class with an __init__ method to initialize name and age.
2. **Define Subclass Student:** Define Student(Person). In its __init__, use super().__init__(name, age) to handle the base attributes, then initialize student_id and major.
3. **Define Subclass Teacher:** Define Teacher(Person). Similarly, use super().__init__(name, age) and then initialize employee_id and subject.
4. **Instantiate Objects:** Create objects for Person, Student, and Teacher with appropriate data.
5. **Test Methods:** Call both inherited methods (introduce) and specialized methods (study_status, teaching_load) to verify functionality.

5. Code

We will use Python for its clear syntax in demonstrating class inheritance.

```
# ----------------------------------------------------------------------
# 5. Code (Person Class Hierarchy)
# This Python code demonstrates inheritance between a base class (Person)
# and two derived classes (Student and Teacher).
# ----------------------------------------------------------------------

class Person:
    """
    Base class representing a general person with common attributes.
    """
    def __init__(self, name: str, age: int):
        # Initialize attributes common to all people
        self.name = name
        self.age = age

    def introduce(self):
        """Prints a basic introduction for the person."""
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

class Student(Person):
    """
    Subclass representing a student, inheriting from Person.
    Adds specialized attributes like student ID and major.
    """
    def __init__(self, name: str, age: int, student_id: str, major: str):
        # Call the constructor of the parent class (Person)
        super().__init__(name, age)
        # Initialize specialized student attributes
        self.student_id = student_id
        self.major = major

    def study_status(self):
        """Prints the student's academic information."""
        print(f"I am a student ({self.student_id}) majoring in {self.major}.")


class Teacher(Person):
    """
    Subclass representing a teacher, inheriting from Person.
    Adds specialized attributes like employee ID and subject.
    """
    def __init__(self, name: str, age: int, employee_id: str, subject: str):
        # Call the constructor of the parent class (Person)
        super().__init__(name, age)
        # Initialize specialized teacher attributes
        self.employee_id = employee_id
        self.subject = subject


    def teaching_load(self):
        """Prints the teacher's professional information."""
        print(f"I am a teacher ({self.employee_id}) specializing in {self.subject}.")


# --- Demonstration ---
```

```python
# 1. Create instances of the classes
print("--- Creating Instances ---")
person1 = Person("Alex Varma", 35)
student1 = Student("Sarah Connor", 20, "S9001", "Computer Science")
teacher1 = Teacher("Dr. Alan Grant", 55, "T402", "Geology")

print("\n--- Testing Person Object ---")
person1.introduce()

print("\n--- Testing Student Object (Inheritance and Specialization) ---")
# Student inherits 'introduce' from Person
student1.introduce()
# Student uses its specialized method
student1.study_status()

print("\n--- Testing Teacher Object (Inheritance and Specialization) ---")
# Teacher inherits 'introduce' from Person
teacher1.introduce()
# Teacher uses its specialized method
teacher1.teaching_load()
```

6. Output

When the `person_hierarchy.py` code is executed, the expected console output will be:

Plaintext

```
--- Creating Instances ---


--- Testing Person Object ---

Hello, my name is Alex Varma and I am 35 years old.


--- Testing Student Object (Inheritance and Specialization) ---

Hello, my name is Sarah Connor and I am 20 years old.

I am a student (S9001) majoring in Computer Science.


--- Testing Teacher Object (Inheritance and Specialization) ---

Hello, my name is Dr. Alan Grant and I am 55 years old.

I am a teacher (T402) specializing in Geology.
```

7. Learning Outcomes

1. **Implementation of Inheritance:** Successfully implemented the core OOP principle of inheritance by defining subclasses (`Student`, `Teacher`) that automatically acquire the attributes and methods of the base class (`Person`).

2. **Parent Constructor Invocation:** Learned the critical use of the `super()` function to correctly call and execute the parent class's `__init__` method from within the subclass's constructor.

3. **Encapsulation and Specialization:** Demonstrated how to maintain common data (encapsulated in `Person`) while adding specialized, unique attributes and behaviors

(methods like `study_status` and `teaching_load`) in the derived classes.

This draft provides a solid foundation. Let me know if you'd like to dive deeper into method overriding, or if you want to implement additional methods in the classes!