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:

Cantillas, Mark G.
BSCS 2A2 Student

"Employee Management System: Employee, Developer, and Manager Classes in Python"

Project Title

PROJECT DESCRIPTION

This code demonstrates the implementation of an Employee Management System using object-oriented programming (OOP) principles in Python. The system consists of three main classes: Employee, Developer, and Manager, each with its own unique attributes and behaviors. The Employee class represents the basic employee, with properties such as name, title, and salary. The display_info() method allows for the easy display of an employee's information, while the give_raise() method enables the adjustment of an employee's salary. The Developer class is a specialized type of Employee, inheriting from the base Employee class. In addition to the basic employee properties, a Developer also has a list of programming languages they are proficient in. The display_info() method for Developer extends the base class implementation to include the list of programming languages. The Manager class is another specialized type of Employee, also inheriting from the base Employee class. A Manager has a list of employees that they manage. The add_employee() method allows for the addition of new employees to the manager's list, and the display_info() method includes the list of employees reporting to the manager.

OBJECTIVES

1. **Define Employee Class:** Create a class to represent employees with attributes like name, title, and salary. Include methods to display employee information and give raises.

2. **Define Developer Class (Inheritance):** Define a subclass of Employee for developers. This subclass should inherit attributes and methods from the Employee class and include an additional attribute for programming languages known by the developer. Override the display_info method to include programming languages.

3. **Define Manager Class (Inheritance):** Define a subclass of Employee for managers. This subclass should inherit attributes and methods from the Employee class and include an additional attribute for a list of employees reporting to the manager. Include a method to add employees to the manager's direct reports. Override the display_info method to include direct reports' information.

4. **Instantiate Objects:** Create instances of the Employee, Developer, and Manager classes with appropriate attributes and relationships.

5. **Display Information:** Call the display_info method for each object to ensure that the information is correctly displayed.

6. **Give Raise to Manager:** Give a raise to the manager object and verify that the salary is updated correctly.

7. **Test Inheritance and Polymorphism**: Ensure that inheritance and polymorphism work as expected by calling methods from both parent and subclass objects.

8. **Ensure Error Handling:** Include error handling where necessary, such as providing default values for optional parameters or checking for valid inputs.

9. **Test Adding Employees to Manager:** Add additional employees to the manager's direct reports and verify that the list is updated correctly.

10. **Ensure Readability and Documentation:** Ensure that the code is well-commented and follows best practices for readability and documentation. This includes using clear variable names, providing comments where necessary, and adhering to PEP 8 style guidelines.

IMPORTANCE AND CONTRIBUTION OF THE PROJECT

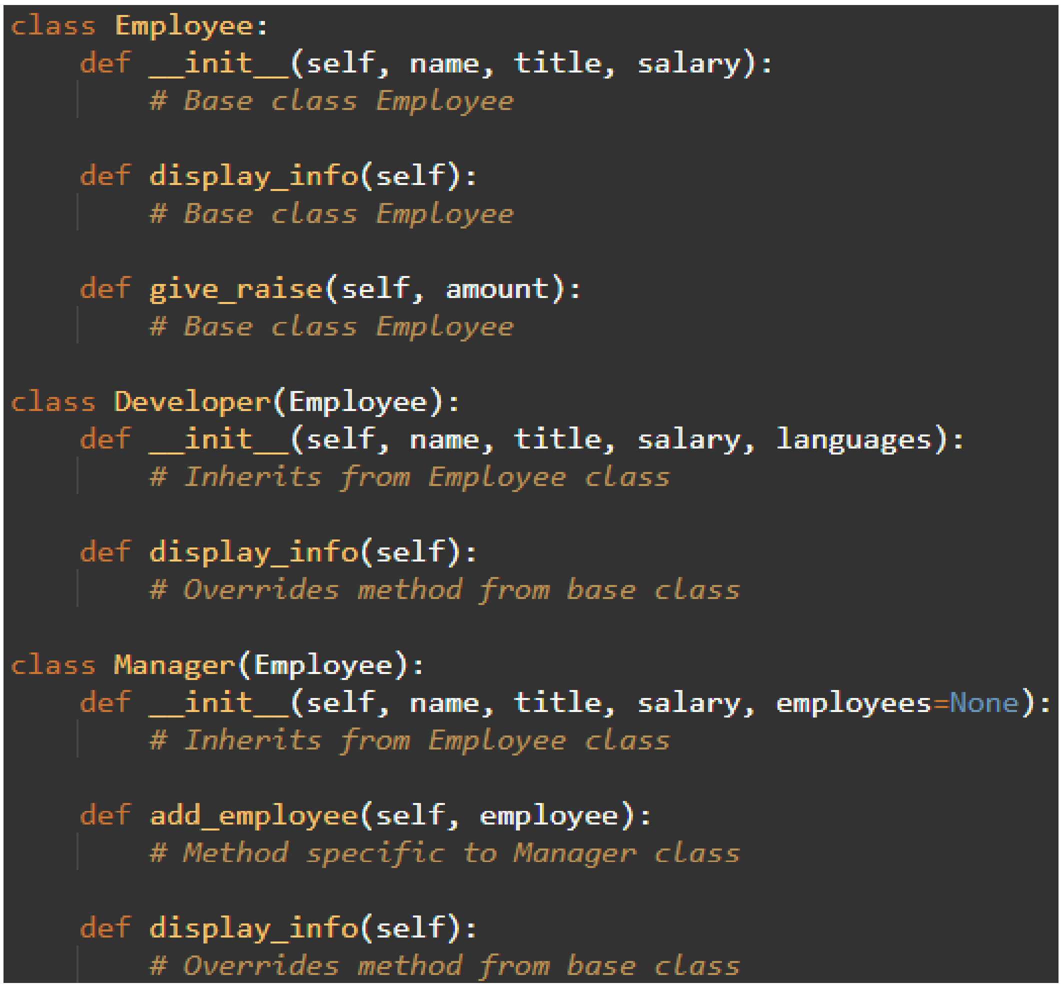
The code represents a simplified model of an employee management system, which can have several implications and contributions in various contexts such as society, school campuses, and among students:

- **Efficient Employee Management:** In a society or organizational setting, efficient employee management is crucial for productivity and organizational success. This code provides a foundation for managing different types of employees, tracking their information, and facilitating communication between managers and their subordinates. Implementing such systems can streamline administrative processes and improve overall workflow efficiency.
- **Educational Tool:** In a school campus or educational setting, this code can serve as an educational tool for teaching object-oriented programming concepts. Students can learn about encapsulation, inheritance, polymorphism, and object instantiation by studying and analyzing the code. Understanding these concepts is essential for students pursuing careers in software development and related fields.
- **Real-world Application:** By studying and analyzing the code, students can gain insights into how object-oriented programming principles are applied in real-world scenarios. This practical knowledge can enhance students' problem-solving skills and prepare them for future roles in software development or IT-related industries.
- **Collaborative Learning:** Working with this code in a classroom or group setting can foster collaborative learning experiences among students. Students can collaborate on assignments or projects involving the modification or extension of the existing codebase. This collaborative approach encourages teamwork, communication, and knowledge sharing among peers.
- **Career Preparation:** Understanding object-oriented programming and software development principles is highly valuable for students pursuing careers in technology-related fields. By working with this code, students can develop foundational skills that are in demand in the job market. Additionally, students can gain practical experience in software development practices such as code organization, documentation, and testing.

FOUR PRINCIPLES OF OOP WITH CODE

ENHERITANCE

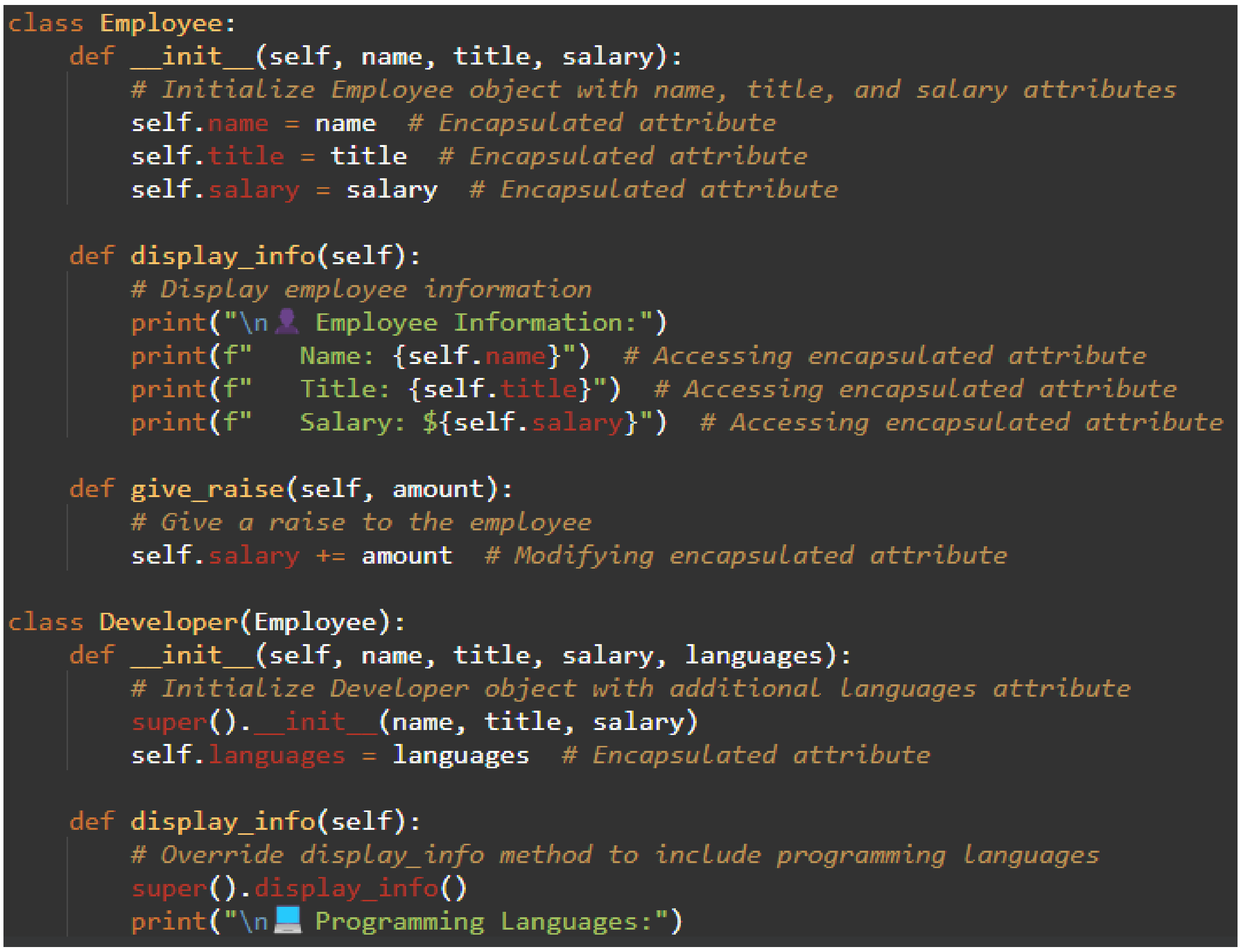
Inheritance is utilized in the Developer and Manager classes, both of which inherit from the Employee class. By using inheritance, the Developer and Manager classes inherit the attributes and methods defined in the Employee class, such as name, title, salary, display_info(), and give_raise(), without the need to redefine them.



```
class Employee:
    def __init__(self, name, title, salary):
        # Base class Employee

    def display_info(self):
        # Base class Employee

    def give_raise(self, amount):
        # Base class Employee

class Developer(Employee):
    def __init__(self, name, title, salary, languages):
        # Inherits from Employee class

    def display_info(self):
        # Overrides method from base class

class Manager(Employee):
    def __init__(self, name, title, salary, employees=None):
        # Inherits from Employee class

    def add_employee(self, employee):
        # Method specific to Manager class

    def display_info(self):
        # Overrides method from base class
```

ENCAPSULATION

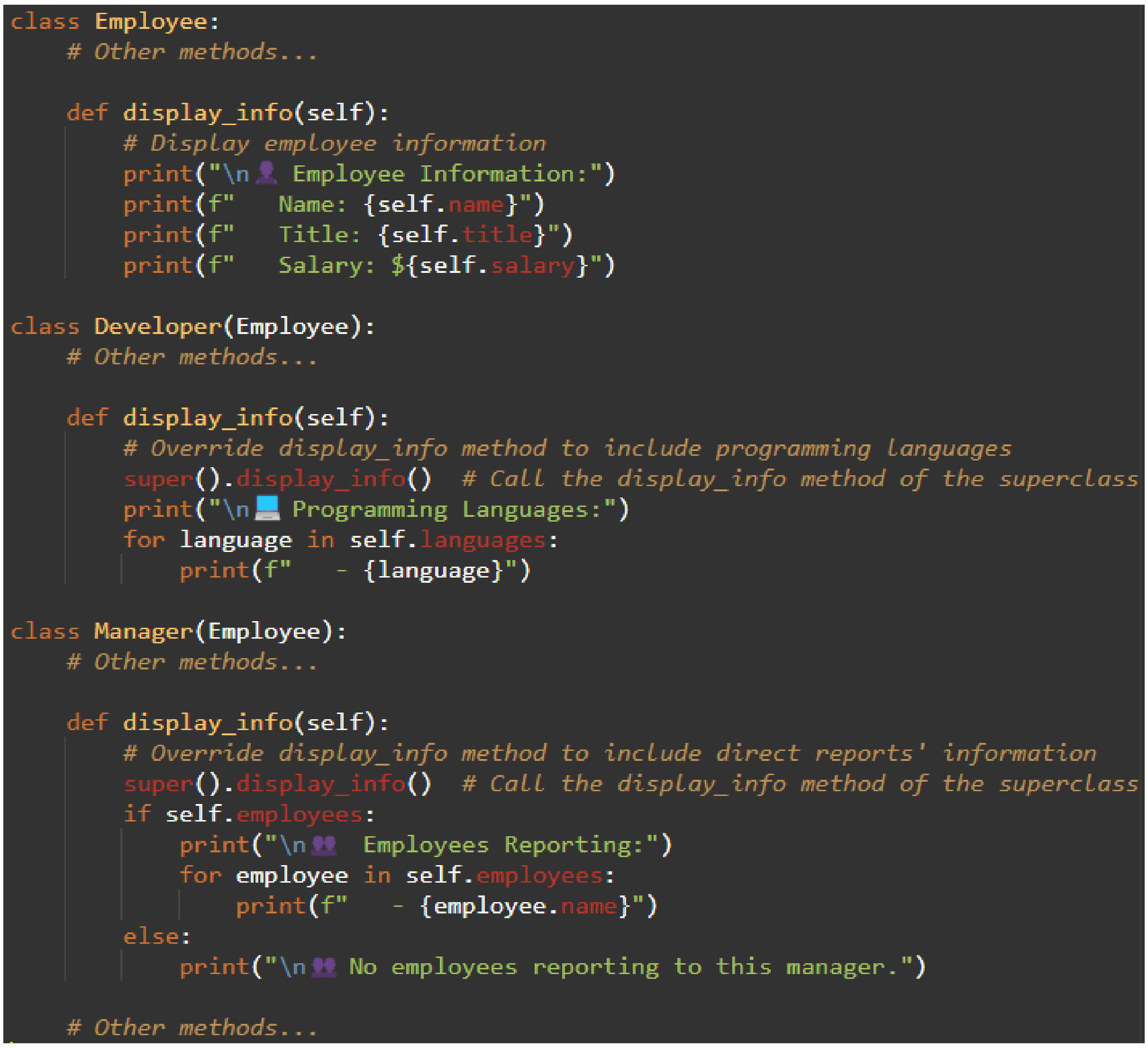
Encapsulation is demonstrated through the use of classes to encapsulate related data (attributes) and behaviors (methods) within objects. For example, the Employee, Developer, and Manager classes encapsulate information about employees, developers, and managers, respectively. Each class defines attributes such as name, title, salary, and methods such as display_info() and give_raise() to operate on the data.



```
class Employee:
    def __init__(self, name, title, salary):
        # Initialize Employee object with name, title, and salary attributes
        self.name = name   # Encapsulated attribute
        self.title = title   # Encapsulated attribute
        self.salary = salary   # Encapsulated attribute

    def display_info(self):
        # Display employee information
        print("\n👤 Employee Information:")
        print(f"   Name: {self.name}")  # Accessing encapsulated attribute
        print(f"   Title: {self.title}")  # Accessing encapsulated attribute
        print(f"   Salary: ${self.salary}")  # Accessing encapsulated attribute

    def give_raise(self, amount):
        # Give a raise to the employee
        self.salary += amount  # Modifying encapsulated attribute

class Developer(Employee):
    def __init__(self, name, title, salary, languages):
        # Initialize Developer object with additional languages attribute
        super().__init__(name, title, salary)
        self.languages = languages  # Encapsulated attribute

    def display_info(self):
        # Override display_info method to include programming languages
        super().display_info()
        print("\n💻 Programming Languages:")
```

POLYMORPHISM

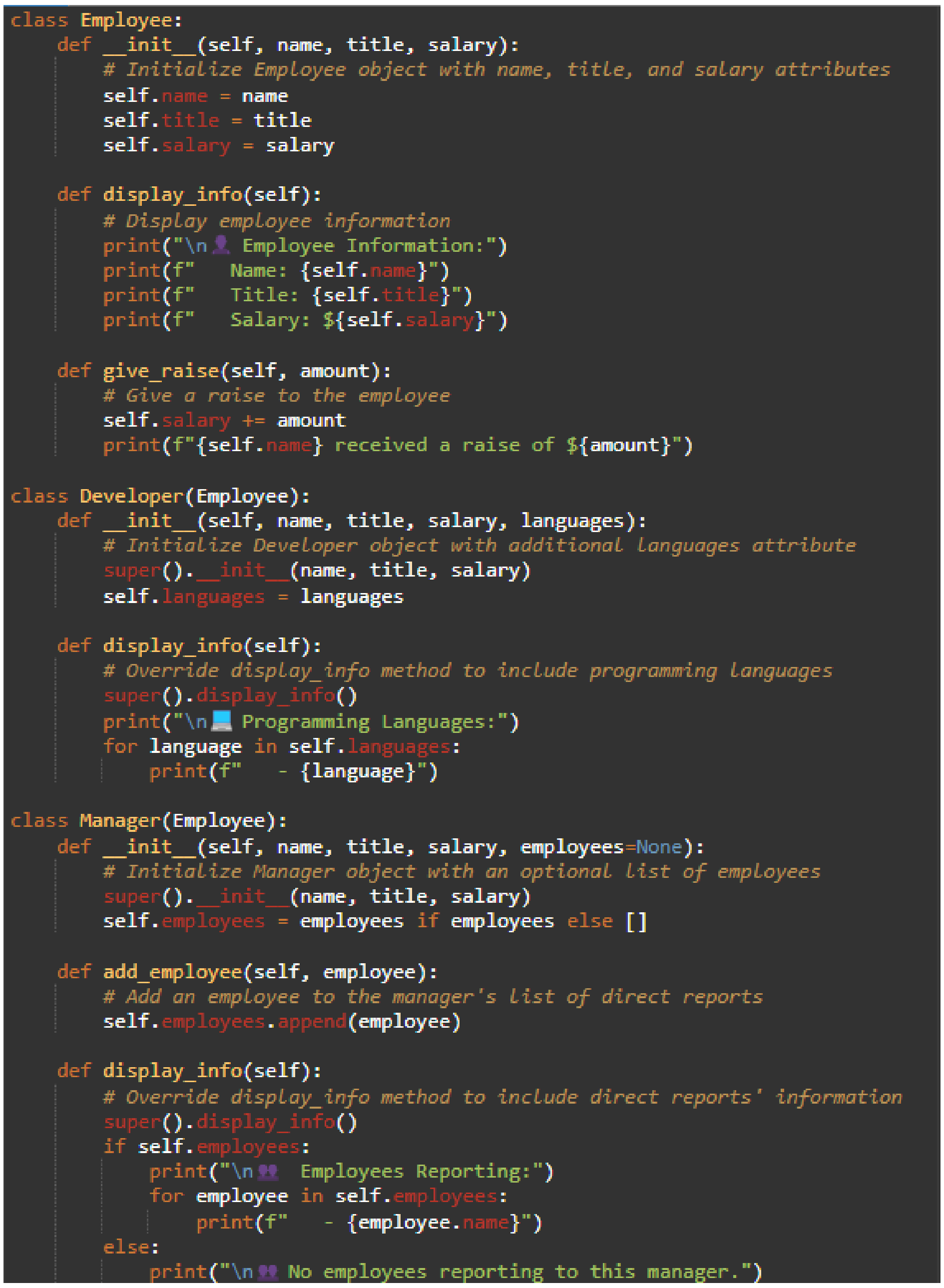
Polymorphism is demonstrated through method overriding in the display_info() method of the Developer and Manager classes. Both subclasses override the display_info() method inherited from the Employee class to provide specialized behavior. Each subclass adds additional information specific to developers (Programming Languages) or managers (Employees Reporting), while still invoking the superclass's display_info() method using super().



```python
class Employee:
    # Other methods...

    def display_info(self):
        # Display employee information
        print("\n👤 Employee Information:")
        print(f"    Name: {self.name}")
        print(f"    Title: {self.title}")
        print(f"    Salary: ${self.salary}")

class Developer(Employee):
    # Other methods...

    def display_info(self):
        # Override display_info method to include programming languages
        super().display_info()  # Call the display_info method of the superclass
        print("\n💻 Programming Languages:")
        for language in self.languages:
            print(f"    - {language}")

class Manager(Employee):
    # Other methods...

    def display_info(self):
        # Override display_info method to include direct reports' information
        super().display_info()  # Call the display_info method of the superclass
        if self.employees:
            print("\n👥  Employees Reporting:")
            for employee in self.employees:
                print(f"    - {employee.name}")
        else:
            print("\n👥 No employees reporting to this manager.")

    # Other methods...
```

ABSTRACTION

Abstraction is applied through the use of classes to model real-world entities (employees, developers, managers) and provide simplified interfaces for interacting with them. Each class encapsulates essential attributes and methods while abstracting away implementation details. Additionally, the code utilizes abstraction to create reusable components, such as the Employee superclass and its subclasses (Developer and Manager), which can be extended and customized as needed.



```python
class Employee:
    def __init__(self, name, title, salary):
        # Initialize Employee object with name, title, and salary attributes
        self.name = name
        self.title = title
        self.salary = salary

    def display_info(self):
        # Display employee information
        print("\n👤 Employee Information:")
        print(f"   Name: {self.name}")
        print(f"   Title: {self.title}")
        print(f"   Salary: ${self.salary}")

    def give_raise(self, amount):
        # Give a raise to the employee
        self.salary += amount
        print(f"{self.name} received a raise of ${amount}")

class Developer(Employee):
    def __init__(self, name, title, salary, languages):
        # Initialize Developer object with additional languages attribute
        super().__init__(name, title, salary)
        self.languages = languages

    def display_info(self):
        # Override display_info method to include programming languages
        super().display_info()
        print("\n💻 Programming Languages:")
        for language in self.languages:
            print(f"   - {language}")

class Manager(Employee):
    def __init__(self, name, title, salary, employees=None):
        # Initialize Manager object with an optional list of employees
        super().__init__(name, title, salary)
        self.employees = employees if employees else []

    def add_employee(self, employee):
        # Add an employee to the manager's list of direct reports
        self.employees.append(employee)

    def display_info(self):
        # Override display_info method to include direct reports' information
        super().display_info()
        if self.employees:
            print("\n👥  Employees Reporting:")
            for employee in self.employees:
                print(f"   - {employee.name}")
        else:
            print("\n👥 No employees reporting to this manager.")
```

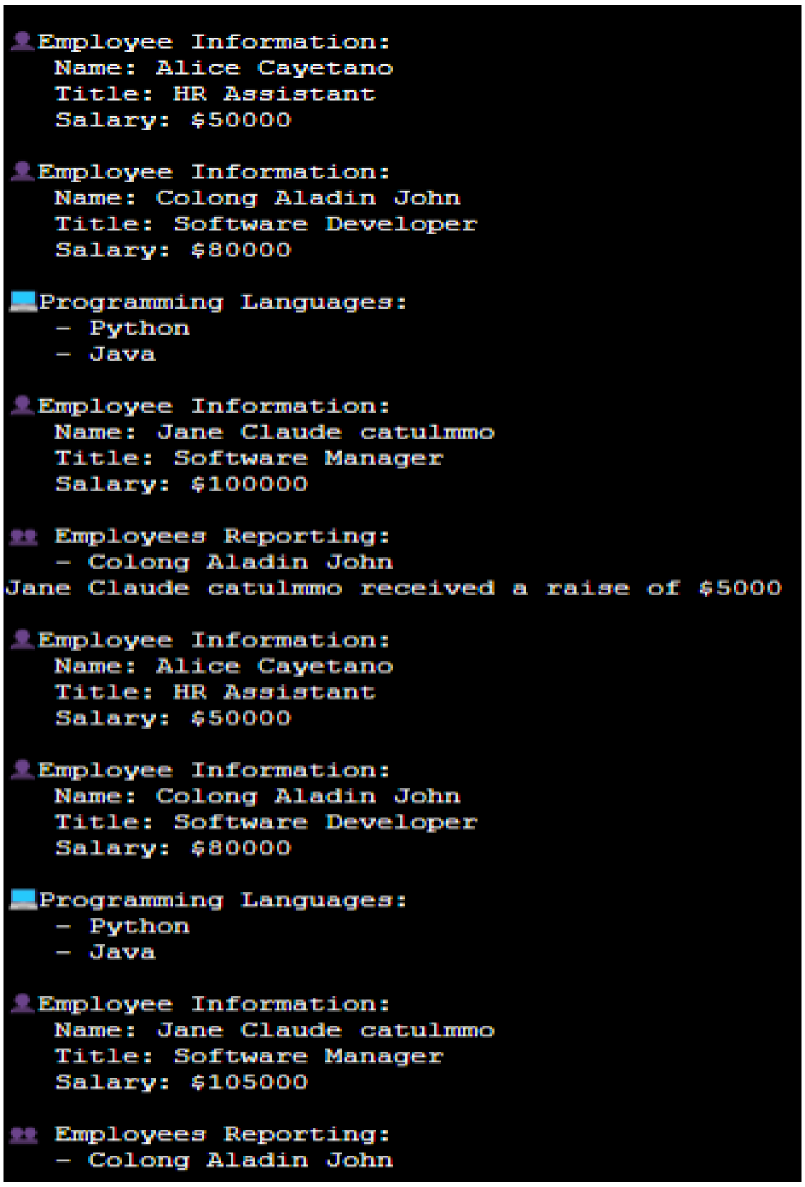
SOFTWARE USED:

- ONLINE GDB

HARDWARE USED:

- SCHOOL PC
- MOBILE PHONE
- CLASSMATE LAPTOP

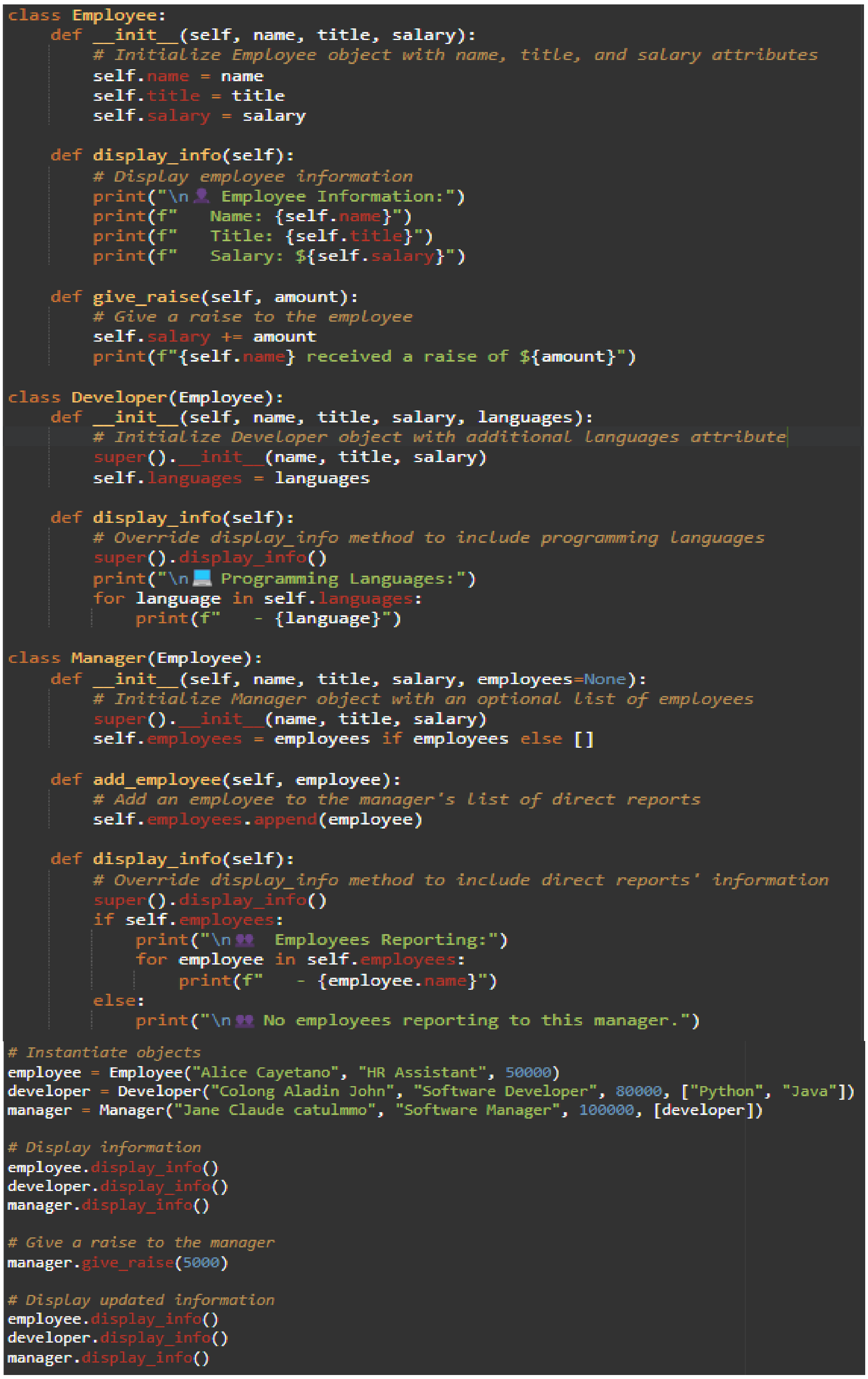
OUTPUT



```
👤Employee Information:
   Name: Alice Cayetano
   Title: HR Assistant
   Salary: $50000

👤Employee Information:
   Name: Colong Aladin John
   Title: Software Developer
   Salary: $80000

💻Programming Languages:
   - Python
   - Java

👤Employee Information:
   Name: Jane Claude catulmmo
   Title: Software Manager
   Salary: $100000

👥 Employees Reporting:
   - Colong Aladin John
Jane Claude catulmmo received a raise of $5000

👤Employee Information:
   Name: Alice Cayetano
   Title: HR Assistant
   Salary: $50000

👤Employee Information:
   Name: Colong Aladin John
   Title: Software Developer
   Salary: $80000

💻Programming Languages:
   - Python
   - Java

👤Employee Information:
   Name: Jane Claude catulmmo
   Title: Software Manager
   Salary: $105000

👥 Employees Reporting:
   - Colong Aladin John
```

Base on the output, there are three employees working in a software development-related field. The first employee, Alice Cayetano, holds the position of HR Assistant with a salary of $50,000. As an HR Assistant, Alice is responsible for managing employee-related tasks, such as recruitment, hiring, onboarding, and handling employee issues. The second employee, Colong Aladin John, is a Software Developer with a salary of $80,000. Colong has a strong background in programming languages, specifically Python and Java. These programming skills enable him to create, test, and maintain software applications, contributing to the development of the company's products and services. The third employee, Jane Claude catulmmo, is a Software Manager with a salary of $105,000. Jane has received a raise of $5,000, which indicates her exceptional performance and contributions to the company. As a Software Manager, Jane oversees the work of other software developers, including Colong Aladin John. Her programming skills in Python and Java allow her to manage and guide her team effectively

CODE



```python
class Employee:
    def __init__(self, name, title, salary):
        # Initialize Employee object with name, title, and salary attributes
        self.name = name
        self.title = title
        self.salary = salary

    def display_info(self):
        # Display employee information
        print("\n👤 Employee Information:")
        print(f"   Name: {self.name}")
        print(f"   Title: {self.title}")
        print(f"   Salary: ${self.salary}")

    def give_raise(self, amount):
        # Give a raise to the employee
        self.salary += amount
        print(f"{self.name} received a raise of ${amount}")

class Developer(Employee):
    def __init__(self, name, title, salary, languages):
        # Initialize Developer object with additional languages attribute
        super().__init__(name, title, salary)
        self.languages = languages

    def display_info(self):
        # Override display_info method to include programming languages
        super().display_info()
        print("\n💻 Programming Languages:")
        for language in self.languages:
            print(f"   - {language}")

class Manager(Employee):
    def __init__(self, name, title, salary, employees=None):
        # Initialize Manager object with an optional list of employees
        super().__init__(name, title, salary)
        self.employees = employees if employees else []

    def add_employee(self, employee):
        # Add an employee to the manager's list of direct reports
        self.employees.append(employee)

    def display_info(self):
        # Override display_info method to include direct reports' information
        super().display_info()
        if self.employees:
            print("\n👥  Employees Reporting:")
            for employee in self.employees:
                print(f"   - {employee.name}")
        else:
            print("\n👥 No employees reporting to this manager.")
# Instantiate objects
employee = Employee("Alice Cayetano", "HR Assistant", 50000)
developer = Developer("Colong Aladin John", "Software Developer", 80000, ["Python", "Java"])
manager = Manager("Jane Claude catulmmo", "Software Manager", 100000, [developer])

# Display information
employee.display_info()
developer.display_info()
manager.display_info()

# Give a raise to the manager
manager.give_raise(5000)

# Display updated information
employee.display_info()
developer.display_info()
manager.display_info()
```

USER GUIDE: EMPLOYEE MANAGEMENT SYSTEM

The Employee Management System is a Python-based application that allows you to manage employee information, including basic details, specialized roles, and reporting structures.

1. Creating an Employee Instance

To create an Employee instance, provide the employee's name, job title, and salary. This instance represents a basic employee within the system.

2. Creating a Developer Instance

To create a Developer instance, include the developer's name, job title, salary, and a list of programming languages they are proficient in. This instance represents a specialized software developer.

3. Creating a Manager Instance and Adding Employees

Create a Manager instance by providing the manager's name, job title, salary, and optionally a list of employees reporting to them. Use the add_employee() method to add employees to the manager's team.

4. Giving a Raise to the Manager

Increase the manager's salary by a specified amount using the give_raise() method. This action demonstrates the system's capability to adjust salaries for employees.

5. Displaying Updated Employee Information

After making changes, display the updated information for all employees, including the basic employee, developer, and manager. This step allows you to view the effects of the actions taken within the system.

REFERENCES

- https://www.onlinegdb.com/
- https://www.w3schools.com/
- https://python.org/
- https://chatgpt.com/