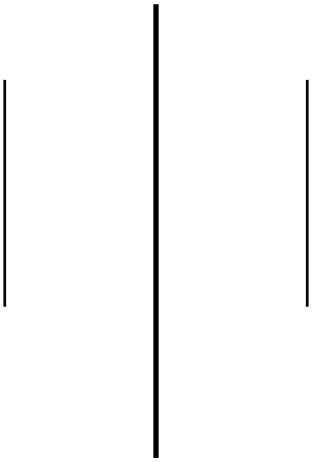
Tribhuvan University

Institute of Science and Technology

Central Department of Computer Science and Information Technology



Object-Oriented Software Engineering
Assignment 1

Submitted by:
Rejina Dahal
Roll No: 15

Submitted to:
Prof. Dr. Subarna Shakya
TU, IOE

Date: March 23, 2025

Assignment Questions and Solutions

Question 1:

Write a program in any OO programming language to demonstrate the use of class, object, instance, inheritance, polymorphism, encapsulation and abstraction.

Solution:

Below is the Python code that implements class, object, instance, inheritance, polymorphism, encapsulation, and abstraction.

Python code:

```
# Base Class
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.__salary = salary

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}, Salary: {
           self._salary}")

    def get_salary(self):
        return self._salary

    def set_salary(self, new_salary):
        if new_salary > 0:
            self._salary = new_salary
        else:
            print("Invalid salary!")


# Derived Classes
class Developer(Employee):
    def __init__(self, name, age, salary, programming_language
       ):
        super().__init__(name, age, salary)
        self.programming_language = programming_language

    def display(self):
        print(f"Developer: {self.name}, Age: {self.age},
           Salary: {self.get_salary()}, Language: {self.
           programming_language}")
```

```
class Manager(Employee):
    def __init__(self, name, age, salary, team_size):
        super().__init__(name, age, salary)
        self.team_size = team_size

    def display(self):
        print(f"Manager: {self.name}, Age: {self.age}, Salary:
            {self.get_salary()}, Team Size: {self.team_size}")


# Creating Instances
emp1 = Employee("Rejina", 22, 50000)
dev1 = Developer("Sam", 30, 70000, "Python")
mgr1 = Manager("Dahal", 35, 80000, 10)

# Displaying Data
emp1.display()
dev1.display()
mgr1.display()

# Accessing Encapsulated Data
print("\nBefore Salary Update:")
print("Rejina's Salary:", emp1.get_salary())

emp1.set_salary(55000)
print("Updated Salary:", emp1.get_salary())
```

Output:

Name: Rejina, Age: 22, Salary: 50000
Developer: Sam, Age: 30, Salary: 70000, Language: Python
Manager: Dahal, Age: 35, Salary: 80000, Team Size: 10
Before Salary Update: Rejina's Salary: 50000
Updated Salary: 55000

Code Explanation with respect to OOP:

Class:

A class is a blueprint or template for creating objects. It defines attributes and behaviors that the objects of that class will have.

- Here, `Employee` is a class that contains attributes like `name`, `age`, and `salary`.
- It also contains methods like `display()`, `get_salary()`, and `set_salary()`.

Object and Instance:

An object is an identifiable entity with data and behaviors. It is a specific instance of a class, containing its own unique data.While an instance is a specific realization or copy of class, a concrete object.

- `emp1` is an object (or instance) of the `Employee` class. It has specific values: "Rejina" as `name`, 22 as `age`, and 50000 as `salary`.
- `dev1` is an instance of `Developer`.
- `mgr1` is an instance of `Manager`.

Inheritance:

Inheritance is a mechanism where a child class or subclass inherits properties and behaviours from another parent or superclass, promoting code reuse and organization.It allows one class to inherit attributes and methods from another class.

- `Developer` inherits from `Employee` using `super().__init__()`, allowing access to `Employee`'s properties and methods.
- Similarly, `Manager` also inherits from `Employee`.

Polymorphism:

Polymorphism is the ability of different objects to respond the same method call in their own way.It allows the same method name to have different behaviors based on the class using it.

- The `display()` method behaves differently in `Employee`, `Developer`, and `Manager`, showing different details.
- When calling `dev1.display()` or `mgr1.display()`, Python automatically chooses the correct method.
- In the program, the `display()` method is overridden in the `Developer` and `Manager` classes.

Information Hiding:

Information Hiding restricts access to the internal details of a class and only exposes necessary functionalities. It ensures **data security** and prevents **unintended modifications**.

- In the above code, the attribute `salary` in the `Employee` class is declared as **private** (using double underscores, `__salary`). This means it should not be accessed directly from outside the class.
- Direct access to `salary` is restricted to **prevent unauthorized modifications** and **maintain encapsulation**.
- Instead, controlled access is provided through the `get_salary()` and `set_salary()` methods, ensuring data integrity.
- The setter method `set_salary()` ensures that salary modifications are only allowed if the new salary is **valid** (e.g., greater than zero).