Practical-16

May 19, 2025

0.1 Practical 16 :-

Name: Sarthak Sanay
Enrollment No: 230031101611051

0.1.1 Problem Statement 1:-

Employee Management System:
Create a base class Employee with public, private, and protected attributes representing employee details such as name, employee ID, and salary. Implement methods to display employee details and calculate salary. Then create subclasses Manager and Developer that inherit from Employee and demonstrate access to different types of attributes and methods.

[8]:
```python
# Base class
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name                # public
        self.__emp_id = emp_id          # private
        self._salary = salary           # protected

    def display_details(self):
        print("Name:", self.name)
        print("Employee ID:", self.__emp_id)
        print("Salary:", self._salary)

    def calculate_salary(self):
        return self._salary

# Manager subclass
class Manager(Employee):
    def __init__(self, name, emp_id, salary, bonus):
        super().__init__(name, emp_id, salary)
        self.bonus = bonus

    def total_salary(self):
        return self._salary + self.bonus  # accessing protected attribute

# Developer subclass
class Developer(Employee):
```

```
    def __init__(self, name, emp_id, salary, project):
        super().__init__(name, emp_id, salary)
        self.project = project

    def show_project(self):
        print(f"{self.name} is working on {self.project}")


m1 = Manager("Raj", "M123", 70000, 10000)
m1.display_details()
print("Total Salary:", m1.total_salary())

print()
d1 = Developer("Dhoni", "D456", 60000, "AI Tool")
d1.display_details()
d1.show_project()
```

```
Name: Raj
Employee ID: M123
Salary: 70000
Total Salary: 80000

Name: Dhoni
Employee ID: D456
Salary: 60000
Dhoni is working on AI Tool
```

0.1.2 Problem Statement 2:-

Bank Account System:
Create a base class Account with public, private, and protected attributes representing account details such as account number, balance, and interest rate. Implement methods to deposit, withdraw, and calculate interest. Then create subclasses SavingsAccount and CheckingAccount that inherit from Account and demonstrate access to different types of attributes and methods.

[9]:
```
# Base class
class Employee:
    def __init__(self, name, emp_id, salary):
        self.name = name                # public
        self.__emp_id = emp_id          # private
        self._salary = salary           # protected

    def display_details(self):
        print("Name:", self.name)
        print("Employee ID:", self.__emp_id)
        print("Salary:", self._salary)

    def calculate_salary(self):
```

```
        return self._salary

# Manager subclass
class Manager(Employee):
    def __init__(self, name, emp_id, salary, bonus):
        super().__init__(name, emp_id, salary)
        self.bonus = bonus

    def total_salary(self):
        return self._salary + self.bonus  # accessing protected attribute

# Developer subclass
class Developer(Employee):
    def __init__(self, name, emp_id, salary, project):
        super().__init__(name, emp_id, salary)
        self.project = project

    def show_project(self):
        print(f"{self.name} is working on {self.project}")


m1 = Manager("Raj", "M123", 70000, 10000)
m1.display_details()
print("Total Salary:", m1.total_salary())

print()
d1 = Developer("Dhoni", "D456", 60000, "AI Tool")
d1.display_details()
d1.show_project()
```

```
Name: Raj
Employee ID: M123
Salary: 70000
Total Salary: 80000

Name: Dhoni
Employee ID: D456
Salary: 60000
Dhoni is working on AI Tool
```

0.1.3 Problem Statement 3:-

Vehicle Management System:
Create a base class Vehicle with public, private, and protected attributes representing vehicle details such as make, model, and year. Implement methods to display vehicle information. Then create subclasses Car and Motorcycle that inherit from Vehicle and demonstrate access to different types of attributes and methods.

[10]:
```python
# Base class
class Vehicle:
    def __init__(self, make, model, year):
        self.make = make              # public
        self._model = model           # protected
        self.__year = year            # private

    def display_info(self):
        print("Make:", self.make)
        print("Model:", self._model)
        print("Year:", self.__year)

# Subclass Car
class Car(Vehicle):
    def __init__(self, make, model, year, doors):
        super().__init__(make, model, year)
        self.doors = doors

    def show_car(self):
        print(f"Car has {self.doors} doors and model is {self._model}")

# Subclass Motorcycle
class Motorcycle(Vehicle):
    def __init__(self, make, model, year, cc):
        super().__init__(make, model, year)
        self.cc = cc

    def show_motorcycle(self):
        print(f"Motorcycle has {self.cc}cc and make is {self.make}")


print("Vehicle Management :-\n")
car1 = Car("Mahindra", "XUV700", 2024, 7)
car1.display_info()
car1.show_car()

print()
bike1 = Motorcycle("Bajaj Motors", "Avenger", 2015, 220)
bike1.display_info()
bike1.show_motorcycle()
```

```
Vehicle Management :-

Make: Mahindra
Model: XUV700
Year: 2024
Car has 7 doors and model is XUV700
```

```
Make: Bajaj Motors
Model: Avenger
Year: 2015
Motorcycle has 220cc and make is Bajaj Motors
```

0.1.4 Problem Statement 4:-

School Management System:
Create a base class Student with public, private, and protected attributes representing student details such as name, roll number, and marks. Implement methods to display student details and calculate grades. Then create subclasses ClassMonitor and Topper that inherit from Student and demonstrate access to different types of attributes and methods.

[11]:
```python
# Base class
class Student:
    def __init__(self, name, roll, marks):
        self.name = name             # public
        self._marks = marks          # protected
        self.__roll = roll           # private

    def display_details(self):
        print("Name:", self.name)
        print("Roll No:", self.__roll)
        print("Marks:", self._marks)

    def calculate_grade(self):
        if self._marks >= 90:
            return "A"
        elif self._marks >= 75:
            return "B"
        elif self._marks >= 60:
            return "C"
        else:
            return "D"

# Subclass ClassMonitor
class ClassMonitor(Student):
    def show_role(self):
        print(f"{self.name} is the class monitor with grade {self.
 ↪calculate_grade()}.")

# Subclass Topper
class Topper(Student):
    def praise(self):
        if self._marks >= 90:
            print(f"{self.name} is the Topper of the class!")
```

```
print("School Management :-\n")
s1 = ClassMonitor("Roshan", 101, 78)
s1.display_details()
s1.show_role()

print()
s2 = Topper("Pankaj", 102, 95)
s2.display_details()
s2.praise()
```

```
School Management :-

Name: Roshan
Roll No: 101
Marks: 78
Roshan is the class monitor with grade B.

Name: Pankaj
Roll No: 102
Marks: 95
Pankaj is the Topper of the class!
```

0.1.5 Problem Statement 5:-

Product Inventory Management:
Create a base class Product with public, private, and protected attributes representing product details such as name, ID, and price. Implement methods to display product information. Then create subclasses ElectronicProduct and ClothingProduct that inherit from Product and demonstrate access to different types of attributes and methods.

[13]:
```
# Base class
class Product:
    def __init__(self, name, pid, price):
        self.name = name            # public
        self._price = price         # protected
        self.__pid = pid            # private

    def show_info(self):
        print("Product Name:", self.name)
        print("Product ID:", self.__pid)
        print("Price:", self._price)

# Subclass ElectronicProduct
class ElectronicProduct(Product):
    def __init__(self, name, pid, price, warranty):
        super().__init__(name, pid, price)
        self.warranty = warranty
```

```
    def show_warranty(self):
        print(f"Warranty: {self.warranty} years")

# Subclass ClothingProduct
class ClothingProduct(Product):
    def __init__(self, name, pid, price, size):
        super().__init__(name, pid, price)
        self.size = size

    def show_size(self):
        print(f"Clothing Size: {self.size}")

print("Product Inventory :-\n")
e1 = ElectronicProduct("Laptop", "E101", 50000, 2)
e1.show_info()
e1.show_warranty()

print()
c1 = ClothingProduct("T-Shirt", "C202", 599, "L")
c1.show_info()
c1.show_size()
```

```
Product Inventory :-

Product Name: Laptop
Product ID: E101
Price: 50000
Warranty: 2 years

Product Name: T-Shirt
Product ID: C202
Price: 599
Clothing Size: L
```