Name:Prabhjot singh

Roll no.: 25901343

Github link- https://github.com/Prabhjot-Singh1/Python-Programming-Laboratory/tree/main/Assignment_5

```
class Student:
    def __init__(self, name, student_id):
        self.name = name
        self.student_id = student_id

    def enroll(self, course):
        course.add_student(self)
        print(f"{self.name} enrolled in {course.course_name}")


class Course:
    def __init__(self, course_name, course_code, department):
        self.course_name = course_name
        self.course_code = course_code
        self.department = department
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def show_students(self):
        print(f"Students enrolled in {self.course_name}:")
        for s in self.students:
            print(f"- {s.name} ({s.student_id})")


class Department:
    def __init__(self, dept_name):
        self.dept_name = dept_name
        self.courses = []

    def add_course(self, course):
        self.courses.append(course)

    def show_courses(self):
        print(f"Courses in {self.dept_name} Department:")
        for c in self.courses:
            print(f"- {c.course_name} ({c.course_code})")



if __name__ == "__main__":

    cs_dept = Department("ARTIFICIAL intelligence")

    python_course = Course("Python Programming", "AI101", cs_dept)
    cs_dept.add_course(python_course)

    s1 = Student("naman", "A001")

    s1.enroll(python_course)

    print()
    cs_dept.show_courses()
    print()
    python_course.show_students()
```

```
naman enrolled in Python Programming

Courses in ARTIFICIAL intelligence Department:
- Python Programming (AI101)

Students enrolled in Python Programming:
- naman (A001)
```

```
class Bill:
    def __init__(self, customer_name, amount):
        self.customer_name = customer_name
        self.amount = amount

    def pay_bill(self):
        raise NotImplementedError("Subclass must implement this method")


class CashPayment(Bill):
    def __init__(self, customer_name, amount, cash_given):
        super().__init__(customer_name, amount)
        self.cash_given = cash_given

    def pay_bill(self):
        if self.cash_given >= self.amount:
            change = self.cash_given - self.amount
            print(f"{self.customer_name} paid ₹{self.amount} in cash.")
            print(f"Change returned: ₹{change}")
        else:
            print("Insufficient cash provided!")


class ChequePayment(Bill):
    def __init__(self, customer_name, amount, cheque_number, bank_name):
        super().__init__(customer_name, amount)
        self.cheque_number = cheque_number
        self.bank_name = bank_name

    def pay_bill(self):
        print(f"{self.customer_name} paid ₹{self.amount} by cheque.")
        print(f"Cheque No: {self.cheque_number}, Bank: {self.bank_name}")


if __name__ == "__main__":
    print("1. Pay by Cash")
    print("2. Pay by Cheque")

    choice = int(input("Enter your choice: "))

    name = input("Enter customer name: ")
    amt = float(input("Enter bill amount: "))

    if choice == 1:
        cash = float(input("Enter cash given: "))
        payment = CashPayment(name, amt, cash)
    else:
        chq_no = input("Enter cheque number: ")
        bank = input("Enter bank name: ")
        payment = ChequePayment(name, amt, chq_no, bank)

    print("\n--- Payment Details ---")
    payment.pay_bill()
```

```
1. Pay by Cash
2. Pay by Cheque
Enter your choice: 2
Enter customer name: harsh
Enter bill amount: 2000
Enter cheque number: 1235409870
Enter bank name: SBI

--- Payment Details ---
harsh paid ₹2000.0 by cheque.
Cheque No: 1235409870, Bank: SBI
```

```
class Distance:
    def __init__(self, meters):
        self.meters = meters

    def __isub__(self, other):
        if isinstance(other, Distance):
            self.meters -= other.meters
            return self
        else:
            raise TypeError("Operand must be a Distance object")

    def __str__(self):
        return f"{self.meters} meters"



d1 = Distance(40)
d2 = Distance(27)

print("Before subtraction:")
print("d1 =", d1)
print("d2 =", d2)

d1 -= d2

print("\nAfter d1 -= d2:")
print("d1 =", d1)
```

```
Before subtraction:
d1 = 40 meters
d2 = 27 meters

After d1 -= d2:
d1 = 13 meters
```

```
class DistanceMeters:
    def __init__(self, meters):
        self.meters = meters

    def to_kilometers(self):
        return self.meters / 1000

    def convert_from_km(self, dist_km):
        print(f"Converting {dist_km.kilometers} km to meters...")
        meters = dist_km.kilometers * 1000
        print(f"{dist_km.kilometers} km = {meters} meters")

    def __str__(self):
        return f"{self.meters} meters"
```

```python
class DistanceKilometers:
    def __init__(self, kilometers):
        self.kilometers = kilometers

    def to_meters(self):
        return self.kilometers * 1000

    def convert_from_m(self, dist_m):
        print(f"Converting {dist_m.meters} meters to kilometers...")
        km = dist_m.meters / 1000
        print(f"{dist_m.meters} meters = {km} km")

    def __str__(self):
        return f"{self.kilometers} kilometers"


d_m = DistanceMeters(750)
d_km = DistanceKilometers(2.5)

print("Distance Objects:")
print(d_m)
print(d_km)
print()

d_m.convert_from_km(d_km)

print()

d_km.convert_from_m(d_m)
```

```
Distance Objects:
750 meters
2.5 kilometers

Converting 2.5 km to meters...
2.5 km = 2500.0 meters

Converting 750 meters to kilometers...
750 meters = 0.75 km
```

Start coding or generate with AI.