Name: Fatima Bint e Naseer

Roll number: SU92-BSAIM-S24-050

Section: 2A

Semester: 2nd

Task = 10

**Program to manage employee personal details:**

**1. Define a parent class called Employee with private attributes name, age, and**

**salary. Implement getter and setter methods for each attribute to ensure controlled**

**access to the data.**

**2. Create a child class Manager inheriting from Employee. The Manager class**

**should have an additional private attribute called department. Implement getter and**

**setter methods for the department attribute.**

**3. Create another child class Worker inheriting from Employee. The Worker class**

**should have an additional private attribute called hours\_worked. Implement getter**

**and setter methods for the hours\_worked attribute.**

**4. Implement file handling to store and retrieve information about employees. Use a**

**CSV file format to store the information in a structured manner, where each row**

**represents an employee and each column represents an attribute (name, age, salary,**

**department, hours\_worked).**

**5. Develop functions to add new employees, display information of all employees,**

**update employee information, and delete employees from the records. Ensure that**

**these functions interact with the Employee class and its subclasses using**

**appropriate encapsulation techniques.**

**6. Provide a user interface to interact with the program, allowing users to perform**

**operations like adding, displaying, updating, and deleting employee information**

**through a menu-driven interface.**

import csv

class Employee:

    def \_\_init\_\_(self, name, age, salary):

        self.\_\_name = name

        self.\_\_age = age

        self.\_\_salary = salary

    def get\_name(self):

        return self.\_\_name

    def set\_name(self, name):

        self.\_\_name = name

    def get\_age(self):

        return self.\_\_age

    def set\_age(self, age):

        self.\_\_age = age

    def get\_salary(self):

        return self.\_\_salary

    def set\_salary(self, salary):

        self.\_\_salary = salary

class Manager(Employee):

    def \_\_init\_\_(self, name, age, salary, department):

        super().\_\_init\_\_(name, age, salary)

        self.\_\_department = department

    def get\_department(self):

        return self.\_\_department

    def set\_department(self, department):

        self.\_\_department = department

class Worker(Employee):

    def \_\_init\_\_(self, name, age, salary, hours\_worked):

        super().\_\_init\_\_(name, age, salary)

        self.\_\_hours\_worked = hours\_worked

    def get\_hours\_worked(self):

        return self.\_\_hours\_worked

    def set\_hours\_worked(self, hours\_worked):

        self.\_\_hours\_worked = hours\_worked

def save\_employees\_to\_file(filename, employees):

    with open(filename, 'w', newline='') as file:

        writer = csv.writer(file)

        writer.writerow(['Name', 'Age', 'Salary', 'Department', 'Hours Worked'])

        for emp in employees:

            if isinstance(emp, Manager):

                writer.writerow([emp.get\_name(), emp.get\_age(), emp.get\_salary(), emp.get\_department(), ''])

            elif isinstance(emp, Worker):

                writer.writerow([emp.get\_name(), emp.get\_age(), emp.get\_salary(), '', emp.get\_hours\_worked()])

def read\_employees\_from\_file(filename):

    employees = []

    try:

        with open(filename, 'r') as file:

            reader = csv.reader(file)

            next(reader)  # Skip header

            for row in reader:

                if row[3]:  # If department is not empty, it's a Manager

                    employees.append(Manager(row[0], int(row[1]), float(row[2]), row[3]))

                elif row[4]:  # If hours\_worked is not empty, it's a Worker

                    employees.append(Worker(row[0], int(row[1]), float(row[2]), int(row[4])))

    except FileNotFoundError:

        print("File not found. Starting with an empty employee list.")

    return employees

def add\_employee(employees):

    emp\_type = input("Enter employee type (Manager/Worker): ").strip().lower()

    name = input("Enter name: ")

    age = int(input("Enter age: "))

    salary = float(input("Enter salary: "))

    if emp\_type == 'manager':

        department = input("Enter department: ")

        new\_employee = Manager(name, age, salary, department)

    elif emp\_type == 'worker':

        hours\_worked = int(input("Enter hours worked: "))

        new\_employee = Worker(name, age, salary, hours\_worked)

    else:

        print("Invalid employee type.")

        return

    employees.append(new\_employee)

    print("Employee added successfully.")

def display\_employees(employees):

    print("\nEmployee List:")

    for emp in employees:

        if isinstance(emp, Manager):

            print(f"Manager: {emp.get\_name()}, Age: {emp.get\_age()}, Salary: {emp.get\_salary()}, Department: {emp.get\_department()}")

        elif isinstance(emp, Worker):

            print(f"Worker: {emp.get\_name()}, Age: {emp.get\_age()}, Salary: {emp.get\_salary()}, Hours Worked: {emp.get\_hours\_worked()}")

def update\_employee(employees):

    name = input("Enter the name of the employee to update: ")

    for emp in employees:

        if emp.get\_name() == name:

            emp.set\_name(input("Enter new name: "))

            emp.set\_age(int(input("Enter new age: ")))

            emp.set\_salary(float(input("Enter new salary: ")))

            if isinstance(emp, Manager):

                emp.set\_department(input("Enter new department: "))

            elif isinstance(emp, Worker):

                emp.set\_hours\_worked(int(input("Enter new hours worked: ")))

            print("Employee updated successfully.")

            return

    print("Employee not found.")

def delete\_employee(employees):

    name = input("Enter the name of the employee to delete: ")

    for emp in employees:

        if emp.get\_name() == name:

            employees.remove(emp)

            print("Employee deleted successfully.")

            return