深 圳 大 学 实 验 报 告

课程名称：现代程序设计

实验项目名称：程序设计综合应用

学院：电子与信息工程学院

专业：

指导教师：邹文斌

报告人：　　学号：　　班级：

实验时间：2023 年 12 月 12 日

实验报告提交时间：2023 年 12 月 25 日

教务部制

一、 实验要求

a) 继承与多态

b) 算法

c) 需上交实验报告、py 源程序。

二、 实验环境

Python IDLE, Pycharm 等

三、 实验内容

运用面向对象的设计思想，开发一个公司员工管理系统。

项目目标：

创建一个功能全面的公司员工管理系统，以便高效地管理员工和经理的信息。

实现员工和经理的增加、编辑、删除功能，以及管理员工之间的关系。

提供查询和显示功能，以便公司能够查看所有员工和经理的信息，或查看特定员工或经理的信息。

实现系统设置功能，以便公司能够设置员工的薪资和职位等信息。

实现数据导入导出功能，以便从外部文件导入员工数据或导出员工数据为外部文件。

项目需求：

员工信息管理：
a. 创建新员工：接收并存储员工的基本信息（例如姓名、职位、薪资等）。
b. 编辑员工信息：实现对现有员工信息的修改。
c. 删除员工：实现从公司员工列表中删除员工的功能。

经理信息管理：
a. 添加经理：接收并存储经理的基本信息（例如姓名、所管理的员工等）。
b. 编辑经理信息：实现对现有经理信息的修改。
c. 删除经理：实现从公司经理列表中删除经理的功能。

员工关系管理：
a. 添加下属：接收并存储经理添加的下属员工信息。
b. 移除下属：删除经理管理的下属员工信息。

查询与显示功能：
a. 显示所有员工的信息：查询并显示所有员工的信息。

b. 显示特定员工的信息：查询并显示特定员工的信息。
c. 显示所有经理的信息：查询并显示所有经理的信息。
d. 显示特定经理的信息：查询并显示特定经理的信息。

其他功能：
a. 系统设置：编写代码来实现设置员工的薪资、职位等信息的功能。
b. 数据导入导出：编写代码来实现从外部文件导入员工数据，或将员工数据导出为外部文件的功能。

技术要求：使用面向对象的设计模式，考虑系统的可扩展性和可维护性。

以下是可能需要实现的一些类、属性和方法：

1. Employee 类：
   属性:
   id: 员工的唯一标识符，整型。
   name: 员工的姓名，字符串类型。
   position: 员工职位，字符串类型。
   salary: 员工薪资，浮点型。
   方法:
   __init__(self, id, name, position, salary): 构造函数，用于初始化员工的属性。
   add_employee(self): 创建一个新员工的实例。
   edit_employee(self, id, name, position, salary): 更新现有员工的信息。
   delete_employee(self, id): 删除员工。
   display_info(self): 用于展示员工的基本信息。

2. EmployeeManager 类：
   属性:
   id: 经理的唯一标识符，整型。
   name: 经理的姓名，字符串类型。
   subordinates: 所管理的员工列表，列表类型。
   方法:
   __init__(self, id, name): 构造函数，用于初始化经理的属性。
   add_subordinate(self, employee_id): 将一个员工添加到另一个经理的管理下。
   remove_subordinate(self, employee_id): 从经理的管理列表中移除一个员工。

3. Manager 类：
   属性:
   id: 经理的唯一标识符，整型。
   name: 经理的姓名，字符串类型。
   方法:
   __init__(self, id, name): 构造函数，用于初始化经理的属性。

4. SystemSettings 类：

属性:
salary_table: 存储员工薪资信息的字典，键为职位，值为薪资。
方法:
set_salary(self, position, salary): 设置一个员工的薪资。

5. DataImporter 类：
属性:
file_path: 指定要导入的文件路径。
方法:
import_data(self): 从指定的文件中导入员工数据。
6. DataExporter 类：
属性:
file_path: 指定要导出的文件路径。
方法:
export_data(self): 将员工数据导出到指定的文件中。

四、 实验过程

思路:

（涉及到算法实现的实验需阐述算法的逻辑关系）

一、基础模块功能

1.按要求编写 Employee 类，设置各项属性，编写构造函数、add_employee、edit_employee、delete_employee、display_info 等方法。同时编写 Manager 类，方法形同 Employee 类。

2.编写 EmployeeManager 类，编写下属管理方法。

3.按要求完成 SystemSettings 类 salary_table 方法和 set_salary 方法的编写。

4.编写文件导入和导出 DataImporter 类和 DataExporter 类。

5.实例化对象，测试代码功能。

二、用户使用设计

1.设计用户使用界面，添加功能按键。

2.设计函数和方法，使功能按键能够调用该方法。

3.设计跳转页面和新页面，使用户体验更好。

4.修改部分函数和方法，使界面设计更合理。

完整代码:

（必须有详细的注释）

一、基础功能 Employee_Manager.py

```
class Employee:
    #存储员工信息
    employee_list = []

    #初始化员工属性
    def __init__(self, id, name, position, salary):
        self.id = id
        self.name = name
        self.position = position
        self.salary = salary
        Employee.employee_list.append(self)

    #创建一个新员工的实例
    def add_employee(self, id, name, position, salary):
        new_employee = Employee(id, name, position, salary)

    #更新现有员工的信息
    def edit_employee(self, id, name, position, salary):
        for employee in Employee.employee_list:
            if employee.id == id:
                employee.name = name
                employee.position = position
                employee.salary = salary

    #删除员工
```

```
    def delete_employee(self, id):
        for employee in Employee.employee_list:
            if employee.id == id:
                Employee.employee_list.remove(employee)

    #用于展示员工的基本信息
    def display_info(self):
        print(f"ID: {self.id}, Name: {self.name}, Position: {self.position},
Salary: {self.salary}")



class SystemSettings:
    #存储员工薪资信息的字典，键为职位，值为薪资
    def __init__(self):
        self.salary_table = {}

    #设置一个员工的薪资
    def set_salary(self, position, salary):
        self.salary_table[position] = salary



class EmployeeManager:
    #初始化经理属性
    def __init__(self, id, name):
        self.id = id
        self.name = name
        self.subordinates = [] #存储经理下属信息

    #将一个员工添加到另一个经理的管理下
```

```
    def add_subordinate(self, employee_id):
        self.subordinates.append(employee_id)

    #从经理的管理列表中移除一个员工
    def remove_subordinate(self, employee_id):
        if employee_id in self.subordinates:
            self.subordinates.remove(employee_id)



class Manager:
    #存储经理信息
    manager_list = []

    #初始化经理的属性
    def __init__(self, id, name):
        self.id = id
        self.name = name
        Manager.manager_list.append(self)

    #创建一个新经理的实例
    def add_manager(self, id, name):
        new_manager = Manager(id, name)

    #更新现有经理的信息
    def edit_manager(self, id, name):
        for manager in Manager.manager_list:
            if manager.id == id:
                manager.name = name
```

```
    #删除经理
    def delete_manager(self, id):
        for manager in Manager.manager_list:
            if manager.id == id:
                Manager.manager_list.remove(manager)

    #用于展示经理的基本信息
    def display_info(self):
        print(f"ID: {self.id}, Name: {self.name}")

class DataImporter:
    # 初始化文件路径
    def __init__(self, file_path):
        self.file_path = file_path

    # 从指定的文件中导入员工数据
    def import_data(self):
        # 实现导入数据的具体逻辑
        print(f"Importing data from {self.file_path}")



class DataExporter:
    # 初始化文件路径
    def __init__(self, file_path):
        self.file_path = file_path

    # 将员工数据导出到指定的文件中
    def export_data(self):
        # 实现导出数据的具体逻辑
```

```
        print(f"Exporting data to {self.file_path}")



# 示例
system = SystemSettings()
system.set_salary("Manager", 50000)
system.set_salary("Assistant", 30000)
system.set_salary("Clerk", 25000)

employee1 = Employee(1, "John", "Manager", 50000)
employee2 = Employee(2, "Alice", "Assistant", 30000)
employee3 = Employee(3, "Bob", "Clerk", 25000)

# 显示所有员工的信息
print("\n显示所有员工的信息：")
for employee in Employee.employee_list:
    employee.display_info()

# 显示特定员工的信息
print("\n显示特定员工的信息：")
employee2.display_info()

# 更新员工信息
employee2.edit_employee(2, "Alice", "Senior Assistant", 35000)
print("\n更新后的员工信息：")
for employee in Employee.employee_list:
    employee.display_info()

# 添加新员工
```

```
employee2.add_employee(4, "Eva", "Clerk", 28000)
print("\n添加新员工后的信息：")
for employee in Employee.employee_list:
    employee.display_info()

# 删除员工
employee2.delete_employee(3)
print("\n删除员工后的信息：")
for employee in Employee.employee_list:
    employee.display_info()

# 实例化经理信息
manager1 = Manager(1, "Michael")
manager2 = Manager(2, "Sarah")

# 显示所有经理的信息
print("\n所有经理的信息：")
for manager in Manager.manager_list:
    manager.display_info()

# 更新经理信息
manager2.edit_manager(2, "Sarah Smith")
print("\n更新后的经理信息：")
for manager in Manager.manager_list:
    manager.display_info()

# 添加新经理
manager3 = Manager(3, "Tom")
print("\n添加新经理后的信息：")
```

```
for manager in Manager.manager_list:
    manager.display_info()

# 删除经理
manager2.delete_manager(2)
print("\n删除经理后的信息：")
for manager in Manager.manager_list:
    manager.display_info()

# 实例化员工信息
employee1 = Employee(1, "John", "Manager", 50000)
employee2 = Employee(2, "Alice", "Assistant", 30000)
employee3 = Employee(3, "Bob", "Clerk", 25000)

# 实例化员工经理关系
employee1_manager = EmployeeManager(1, "Michael")
employee2_manager = EmployeeManager(2, "Sarah")
employee3_manager = EmployeeManager(1, "Michael")

# 添加下属
employee2_manager.add_subordinate(2)
employee2_manager.add_subordinate(3)

# 显示经理的下属
print("\n" + f"{manager1.name}的下属：{employee1.name}")
print("\n" + f"{manager2.name}的下属：{employee2.name}, {employee3.name}")

# 移除下属
employee2_manager.remove_subordinate(3)
```

```
# 显示经理的下属
print(f"\n{manager2.name}的下属：{employee2.name}\n")



importer = DataImporter(None)
importer.import_data()

exporter = DataExporter(None)
exporter.export_data()
```

二、用户界面 **Management_System.py**

（只展示 EmployeeManagerApp 类的部分代码，完整代码在附件源程序中）

```
    #用于添加员工或经理
    def add_employee_or_manager(self):
        # 获取职位、姓名和薪水输入框中的值
        position = self.position_var.get()
        name = self.name_entry.get()
        salary = self.salary_entry.get()
        # 如果职位是员工
        if position == "员工":
            # 创建一个新的员工对象
            new_employee = Employee(len(Employee.employee_list) + 1, name, "
员工", salary)
            # 在员工列表框中插入员工的姓名
            self.employee_listbox.insert(tk.END, name)
            # 清空姓名和薪水输入框
            self.name_entry.delete(0, tk.END)
            self.salary_entry.delete(0, tk.END)
```

```
            print(f"Added new employee: {name}")
        # 如果职位是经理
        elif position == "经理":
            # 创建一个新的经理对象
            new_manager = Manager(len(Manager.manager_list) + 1, name,
position, salary)
            # 在经理列表框中插入经理的姓名
            self.manager_listbox.insert(tk.END, name)
            # 清空姓名和薪水输入框
            self.name_entry.delete(0, tk.END)
            self.salary_entry.delete(0, tk.END)
            print(f"Added new manager: {name}")

    #用于删除选定员工
    def delete_employee(self):
        # 获取当前选中的员工在员工列表框中的索引
        selected_index = self.employee_listbox.curselection()
        # 如果有选中的员工
        if selected_index:
            # 弹出确认对话框，确认是否删除选中的员工
            confirm = messagebox.askokcancel("确认删除", "确定要删除该员工
吗？")
            # 如果确认删除
            if confirm:
                # 获取选中员工的姓名
                employee_name = self.employee_listbox.get(selected_index)
                # 遍历员工列表，找到对应姓名的员工并删除
                for employee in Employee.employee_list:
                    if employee.name == employee_name:
```

```
                        Employee.employee_list.remove(employee)
                        break
                # 从员工列表框中删除选中的员工
                self.employee_listbox.delete(selected_index)
    #用于查看员工信息
    def view_employee_info(self):
        # 获取当前选中的员工在员工列表框中的索引
        selected_index = self.employee_listbox.curselection()
        # 如果有选中的员工
        if selected_index:
            # 获取选中员工的姓名和 ID
            employee_name = self.employee_listbox.get(selected_index)
            employee_id = selected_index[0] + 1
            # 遍历员工列表，找到对应的员工
            for employee in Employee.employee_list:
                if employee.name == employee_name and employee.id ==
selected_index[0] + 1:
                    # 调用显示信息窗口方法，显示员工的信息
                    self.show_info_window(employee.display_info())

    # 用于查看经理信息
    def view_manager_info(self):
        # 获取当前选中的经理在经理列表框中的索引
        selected_index = self.manager_listbox.curselection()
        # 如果有选中的经理
        if selected_index:
            # 获取选中经理的 ID
            manager_id = selected_index[0] + 1
            # 遍历经理列表，找到对应 ID 的经理
```

```
            for manager in Manager.manager_list:
                if manager.id == manager_id:
                    # 调用显示经理信息窗口方法，显示经理的信息
                    self.show_manager_info_window(manager)

    # 为经理添加下属
    def add_subordinate_to_manager(self, manager, subordinate_combobox, subordinate_listbox):
        # 获取选定的员工
        selected_subordinate = subordinate_combobox.get()
        # 检查员工是否已经在经理的管理下
        if any(employee.name == selected_subordinate for employee in manager.subordinates):
            messagebox.showerror("错误", f"员工 {selected_subordinate} 已经在经理 {manager.name} 的管理下")
        else:
            # 遍历员工列表，查找选定的下属
            for employee in Employee.employee_list:
                if employee.name == selected_subordinate:
                    # 添加员工对象到经理的下属列表
                    manager.add_subordinate(employee)
                    # 在下属列表框中插入员工的姓名
                    subordinate_listbox.insert(tk.END, selected_subordinate)
                    messagebox.showinfo("成功", f"员工 {selected_subordinate} 已成功添加到经理 {manager.name} 的管理下")
                    manager_info_window = subordinate_listbox.master
                    manager_info_window.destroy()  # 关闭当前对话框
                    self.show_manager_info_window(manager)  # 重新打开经理信息的页面
```

```
                    break
            else:
                messagebox.showerror(" 错 误 ",  f" 未 找 到 名 为 {selected_subordinate} 的员工")

    # 从经理的下属列表中删除员工
    def delete_subordinate(self, manager, subordinate_listbox):
        selected_subordinate = subordinate_listbox.get(subordinate_listbox.curselection())
        # 遍历员工列表，查找选定的下属
        for employee in Employee.employee_list:
            if employee.name == selected_subordinate:
                manager.remove_subordinate(employee)  # 从下属列表中移除该员工对象
                messagebox.showinfo("成功", f"员工 {selected_subordinate} 已成功从经理 {manager.name} 的管理下移除")
                # 从下属列表框中删除员工的姓名

subordinate_listbox.delete(subordinate_listbox.curselection())
                manager_info_window = subordinate_listbox.master
                manager_info_window.destroy()  # 关闭当前对话框
                self.show_manager_info_window(manager)  # 重新打开经理信息的页面
                return
        messagebox.showerror("错误", f"未找到名为 {selected_subordinate} 的员工")
    # 更新员工或经理信息
    def update_employee_or_manager_info(self):
        selected_index_employee = self.employee_listbox.curselection()
```

```
        selected_index_manager = self.manager_listbox.curselection()
        # 选中员工
        if selected_index_employee:
            employee_name = self.employee_listbox.get(selected_index_employee)
            for employee in Employee.employee_list:
                if employee.name == employee_name:
                    self.update_info_dialog(employee)  # 调用更新信息对话框
                    break
        # 选中经理
        elif selected_index_manager:
            manager_name = self.manager_listbox.get(selected_index_manager)
            for manager in Manager.manager_list:
                if manager.name == manager_name:
                    self.update_info_dialog(manager)
                    break
        # 从文件导入数据
    def import_data_from_file(self):
        file_path = filedialog.askopenfilename()  # 使用文件对话框选择文件
        if file_path:
            data_importer = DataImporter(file_path)  # 创建数据导入器对象
            data_importer.import_data()  # 导入数据
            messagebox.showinfo("成功", f"从文件 {file_path} 中成功导入员工数据")

    # 导出数据到文件
    def export_data_to_file(self):
        file_path = filedialog.asksaveasfilename()  # 使用文件对话框选择保存文件的路径
```

```
        if file_path:
            data_exporter = DataExporter(file_path) # 创建数据导出器对象
            data_exporter.export_data() # 导出数据
            messagebox.showinfo("成功", f"成功将员工数据导出到文件{file_path}")
```

五、 实验结果

（运行结果，截图）

一、基础功能 Employee_Manager.py

```
显示所有员工的信息:
ID: 1, Name: John, Position: Manager, Salary: 50000
ID: 2, Name: Alice, Position: Assistant, Salary: 30000
ID: 3, Name: Bob, Position: Clerk, Salary: 25000

显示特定员工的信息:
ID: 2, Name: Alice, Position: Assistant, Salary: 30000

更新后的员工信息:
ID: 1, Name: John, Position: Manager, Salary: 50000
ID: 2, Name: Alice, Position: Senior Assistant, Salary: 35000
ID: 3, Name: Bob, Position: Clerk, Salary: 25000

添加新员工后的信息:
ID: 1, Name: John, Position: Manager, Salary: 50000
ID: 2, Name: Alice, Position: Senior Assistant, Salary: 35000
ID: 3, Name: Bob, Position: Clerk, Salary: 25000
ID: 4, Name: Eva, Position: Clerk, Salary: 28000

删除员工后的信息:
ID: 1, Name: John, Position: Manager, Salary: 50000
ID: 2, Name: Alice, Position: Senior Assistant, Salary: 35000
ID: 4, Name: Eva, Position: Clerk, Salary: 28000

所有经理的信息:
ID: 1, Name: Michael
ID: 2, Name: Sarah

更新后的经理信息:
ID: 1, Name: Michael
ID: 2, Name: Sarah Smith

添加新经理后的信息:
ID: 1, Name: Michael
ID: 2, Name: Sarah Smith
ID: 3, Name: Tom

删除经理后的信息:
ID: 1, Name: Michael
ID: 3, Name: Tom

Michael的下属: John

Sarah Smith的下属: Alice, Bob

Sarah Smith的下属: Alice

Importing data from None
Exporting data to None
```

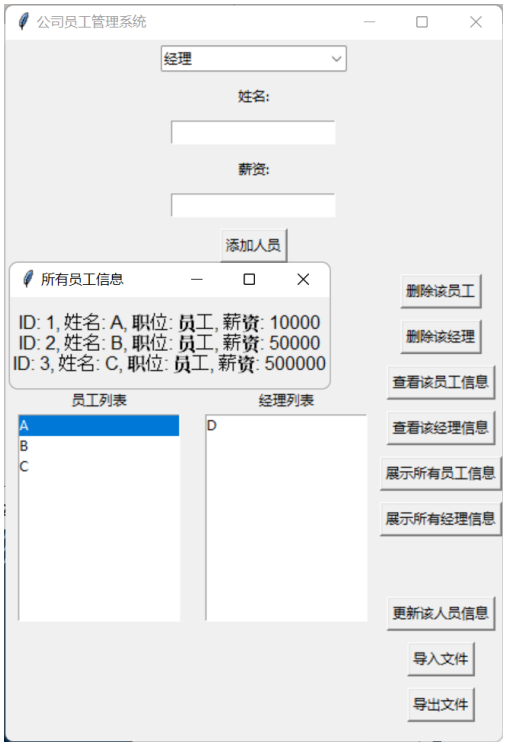
二、添加用户使用模块 Management_System.py

1.主页面

2.使用添加员工/经理功能

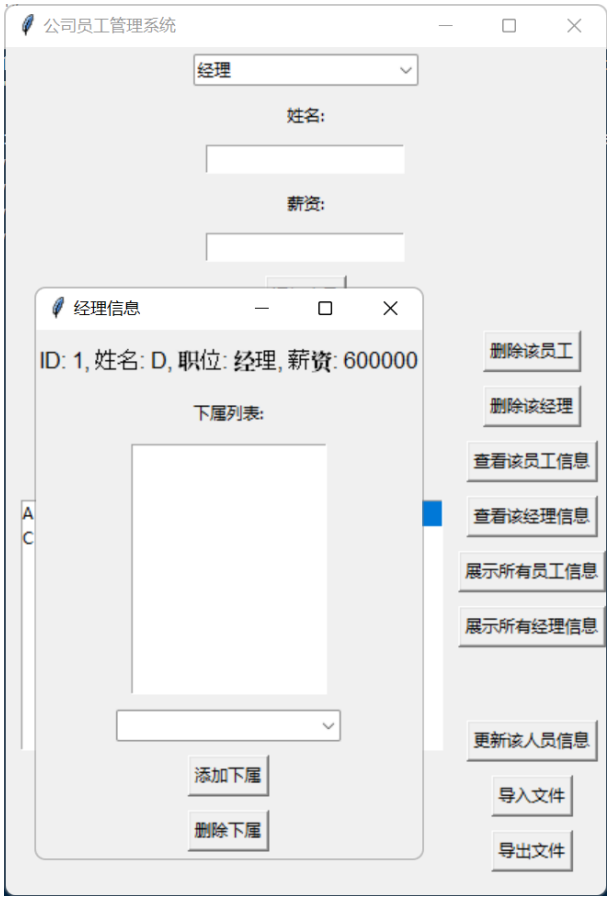
3.查看特定员工/经理信息功能

4.查看全部员工/经理信息功能



5.删除选定员工/经理功能

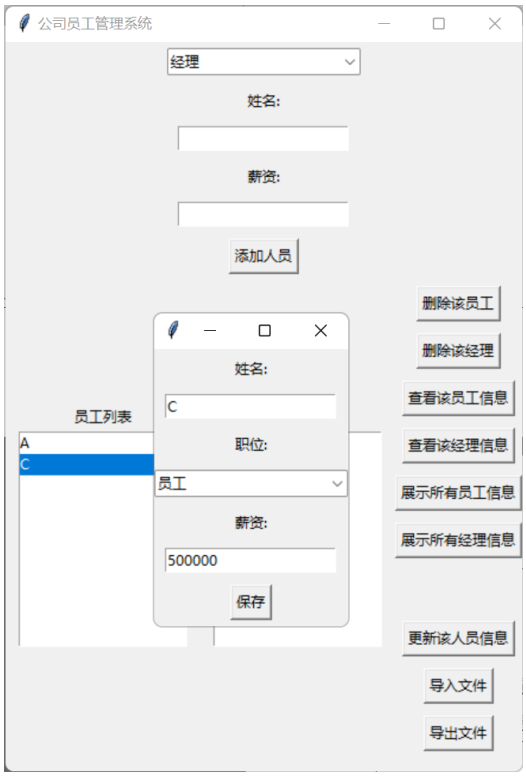
6.查看选定经理信息功能

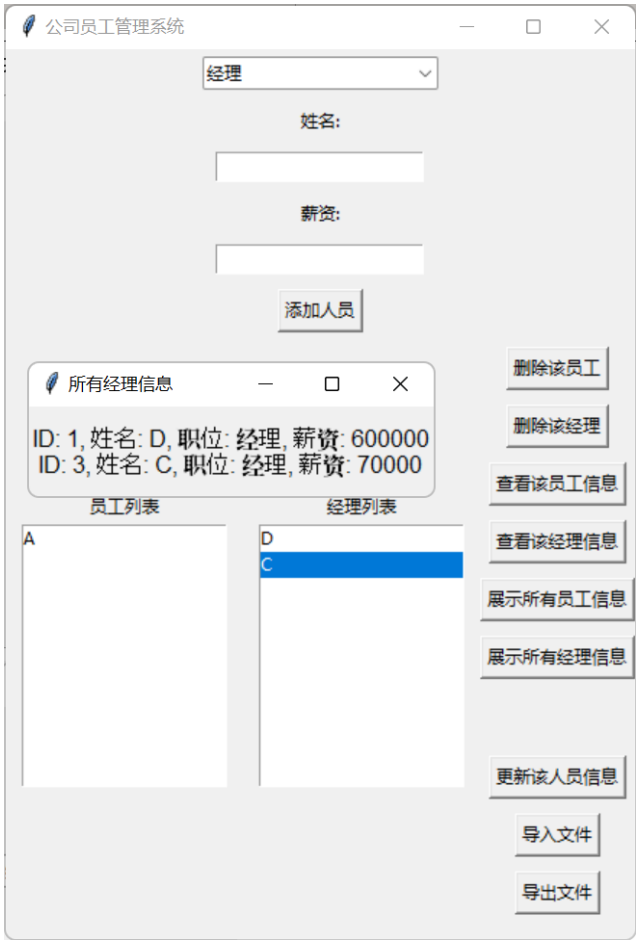


7.为经理添加下属功能

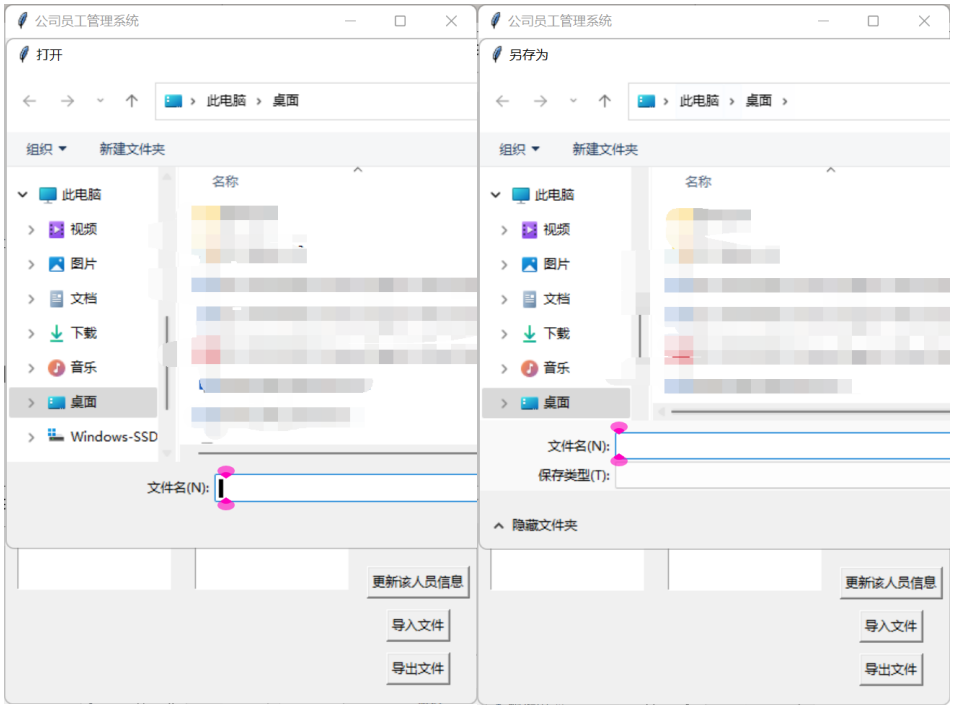
8.删除下属功能

9.更新选定人员信息功能





10.文件导入与导出功能



六、 实验心得

（本次实验遇到的问题，解决过程，有什么收获等）

1.搜查资料学习了 tkinter 库。

2.学习了用 tkinter 库设置用户界面、文本框、下拉框、功能按键等内容。

3.学习了怎样写函数方法连接设置的按键和实现功能。

4.初步学会了编写用户使用界面。

成绩评定：

实验过程（60 分）	实验结果（30 分）	心得体会（10 分）	总分（100 分）

指导教师签字：　　年　月　日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。