Báo cáo

Khoa công nghệ thông tin

Trường đại học Khoa học tự nhiên

Đại học quốc gia thành phố Hồ Chí Minh

Thông tin sinh viên

> ***Họ và tên*: Thái Nguyễn Việt Hùng**
>
> ***Mã số sinh viên*: 20120488**
>
> ***Lớp*: Thực hành OOP nhóm 3 20_1**

Các phần đã làm được

> **Tìm hiểu về design pattern**
>
> **Tìm hiểu và thực hành Singleton, Factory, Prototype, Builder**
>
> **Áp dụng Singleton, Factory vào dự án.**

Cách biên dịch và chạy dự án

Mô tả lại dự án

Tính và xuất ra lương của nhân viên theo những loại nhân viên khác nhau:

> - Nhân viên tính lương theo ngày
> - Nhân viên tính lương theo giờ
> - Nhân viên tính lương theo sản phẩm
> - Quản lí: Tính theo lương cứng, số lượng nhân viên quản lí

Các class và các hàm quan trọng

Class nhân viên

```
class Employee {
protected:
 string fullname;
 float Salary;
public:
 virtual int get_Total() {
  return 0;
 }
 virtual void set_Total(int) {
  return;
 }
 virtual float get_Payment() {
  return 0;
 }
 virtual void set_Payment(float k) {
  return;
 }
 virtual float get_fixed() {
  return 0;
 }
 virtual void set_fixed(float) {
  return;
 }
 virtual float countSalary() {
  return 0;
 }
 virtual int Tick() {
  return 0;
 }

 static Employee* make_Employee(int choice);
 string get_fullname();
 void set_fullname(string);
};
```

Class nhân viên tính lương theo ngày

```
class DailyEmployee : public Employee {
private:
 int days;
 float DailyPayment;
public:
 int get_Total();
 void set_Total(int);
 float get_Payment();
 void set_Payment(float k);
 float countSalary();
 int Tick() {
  return 1;
 }
};
```

Class nhân viên tính lương theo giờ

```
class HourlyEmployee :public Employee {
private:
 int hours;
 float HourlyPayment;
public:
 int get_Total();
 void set_Total(int);
 float get_Payment();
 void set_Payment(float);
 float countSalary();
 int Tick() {
  return 2;
 }
};
```

Class nhân viên tính lương theo sản phẩm

```
class ProductEmployee : public Employee {
private:
 int products;
 float PaymentPerProduct;

public:
 int get_Total();
 void set_Total(int);
 float get_Payment();
 void set_Payment(float);
 float countSalary();
 int Tick() {
  return 3;
 }
};
```

Class quản lí

```
class Manager : public Employee {
private:
 float fixedPayment;
 int TotalEmployees;
 float PaymentPerEmployee;
public:
 float get_fixed();
 int get_Total();
 float get_Payment();
 void set_fixed(float);
 void set_Total(int);
 void set_Payment(float);
 float countSalary();
 int Tick() {
  return 0;
 }
};
```

Theo mô tả trên, các class *Employee*; *DailyEmployee*, *HourlyEmployee*, P*roductEmployee*, Manager đều được thừa kế từ class Nhân viên. Vì vậy thay vì viết những phương thức khởi tạo cho từng loại nhân viên, ta sẽ viết chung 1 phương thức khởi tạo cho Nhân viên, kiểu trả về từng loại nhân viên phụ thuộc vào tham số truyền vào. Cụ thể:

```
Employee* Employee::make_Employee(int choice) {
 if (choice == 0)
  return new Manager;
 if (choice == 1)
  return new DailyEmployee;
 if (choice == 2)
  return new HourlyEmployee;
 if (choice == 3)
  return new ProductEmployee;
}
```

> □ Đây chính là Factory method pattern

Ngoài ra, ở đây chúng ta tạo thêm một Class để quản lí dữ liệu, nó sẽ quản lí tất cả dữ liệu của nhân viên. Cụ thể:

```
class DataManeger {
private:
 string name;
 vector<Employee*> arr;
 static DataManeger* mInstancePtr;
 static mutex mLocker;
 DataManeger(string x) {
  name = x;
 }
public:
 DataManeger(const DataManeger&) = delete;
 static DataManeger* getInstance(string x);
 void readFile(const char* Input);
 void countAllSalary();
 void printInfo();
};
```

```
DataManeger* DataManeger::mInstancePtr = nullptr;
mutex DataManeger::mLocker;


DataManeger* DataManeger::getInstance(string x) {
 mLocker.lock();
 if (nullptr == mInstancePtr) {
  mInstancePtr = new DataManeger(x);
 }
 mLocker.unlock();
 return mInstancePtr;
}
```

Ở class *DataManeger*, thay vì để hàm khởi tạo ngoài public, ta sẽ để nó trong private. Mong muốn của chúng ta là chỉ tạo duy nhất một đối tượng *DataManeger*, vì vậy chúng ta cần tạo một static method có tên là *getIntance()* dùng để tạo đối tượng của class *DataManeger*. Method nãy sẽ gọi đến constructor của *DataManeger*. Hàm này cũng chịu trách nhiệm đảm

bảo chỉ có duy nhất một object của class được tạo ra trong hệ thống.

Ngoài ra, để tránh có nhiều threads cùng call đến hàm *getInstance()* cùng lúc. Ta dùng mutex để đồng bộ xử lí của các threads.

> □ Đây là Singleton Pattern

Cách thức hoạt động

Chương trình sẽ đọc dữ liệu từ file *November2021.txt**** . Sau đó sẽ thực hiện tính toán lương theo công thức của từng loại nhân viên và xuất ra console.

Demo

Youtube_linkdemo