Sistema de Gestión de Empresa

Table of Contents

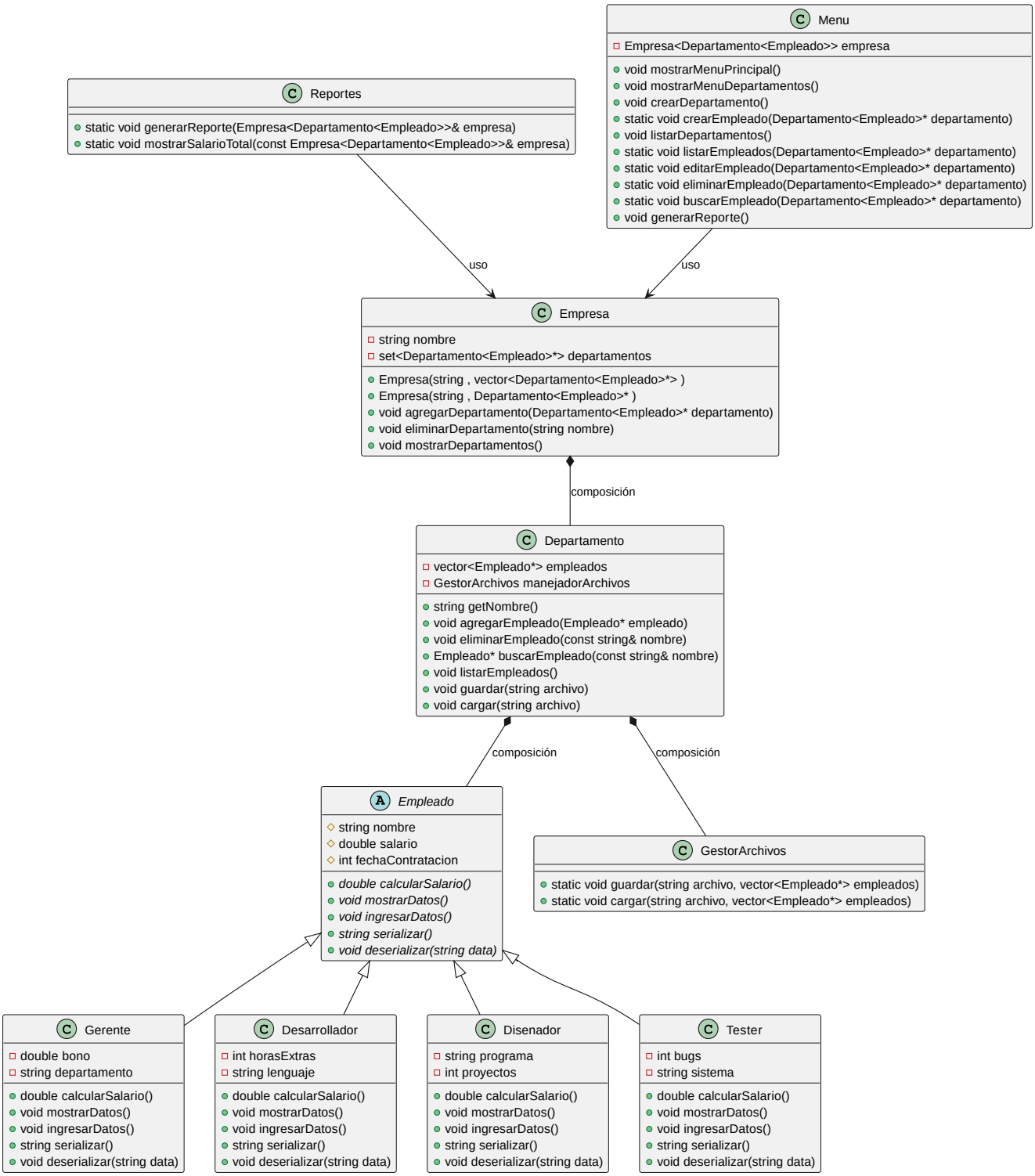
1. Antecedentes

Este proyecto tiene como objetivo gestionar la información de empleados, departamentos y reportes dentro de una empresa. Proporciona una interfaz de usuario a través de un menú para interactuar con el sistema.

2. Requisitos

- Debe permitir gestionar la información de los empleados.
- Debe permitir la generación de reportes.
- Debe ser fácil de usar a través de un menú interactivo.
- Debe almacenar los datos de forma persistente.

3. Diagrama de Clases



4. Método

4.1. Estructura del Proyecto

El proyecto está estructurado en varios archivos que representan diferentes componentes del sistema:

- `main.cpp`: Punto de entrada del programa.
- `Empresa.h`: Gestión de la información de la empresa.

- `Reporte.h`: Generación de reportes.
- `Contenedor.h`: Contenedor de datos de empleados.
- `menu.h`: Interfaz de usuario mediante menú.
- `Empleados/`: Contiene varias clases de empleados como `Desarrollador`, `Diseñador`, `Empleado`, `Gerente`, y `Tester`.

4.2. Explicación del Código

4.2.1. main.cpp

```
#include "src/Empresa.h"
#include "src/Reporte.h"
#include "src/Contenedor.h"   // Es un archivo que contiene los datos de los empleados
#include "src/Menu.h"
using namespace std;
int main() {

    Menu menu;
    menu.mostrarMenuPrincipal();
    return 0;

}
```

El archivo `main.cpp` inicializa el menú principal del sistema y espera interacciones del usuario.

4.2.2. Empresa.h

```
#ifndef TRABAJOS_LP_2_EMPRESA_H
#define TRABAJOS_LP_2_EMPRESA_H

#include <set>
#include "Departamento.h"

template <typename T>
class Empresa {
public:
    explicit Empresa(string nombre, vector<T*> departamento) : nombre(nombre)
,departamentos(departamento) {
    }

    Empresa(string nombre, T* departamento) : nombre(nombre) {
        departamentos.insert(departamento);
    }

    ~Empresa() {
        for (auto departamento : departamentos) {
```

```
            delete departamento;
        }
    }

    void agregarDepartamento(T* departamento) {
        departamentos.insert(departamento);
    }

    void eliminarDepartamento(const string& nombre) {
        departamentos.erase(remove_if(departamentos.begin(), departamentos.end(),
                                      [&nombre](T* departamento) {
                                          return departamento->getNombre() == nombre;
                                      }), departamentos.end());
        cout << "Departamento eliminado correctamente" << endl;
    }

    void mostrarDepartamentos() const {
        for (const auto& departamento : departamentos) {
            cout << departamento->getNombre() << endl;
        }
    }

private:
    string nombre;
    set<T*> departamentos;
};

#endif //TRABAJOS_LP_2_EMPRESA_H
```

La clase `Empresa` es una plantilla que maneja una colección de departamentos.

- **Constructores**:
- `Empresa(string nombre, vector<T*> departamento)`: Inicializa la empresa con un nombre y un vector de departamentos.
- `Empresa(string nombre, T* departamento)`: Inicializa la empresa con un nombre y un único departamento.
- **Destructor**:
- `~Empresa()`: Libera la memoria de los departamentos cuando se destruye el objeto `Empresa`.
- **Métodos**:
- `void agregarDepartamento(T* departamento)`: Añade un nuevo departamento a la empresa.
- `void eliminarDepartamento(const string& nombre)`: Elimina un departamento por nombre.
- `void mostrarDepartamentos() const`: Muestra los nombres de todos los departamentos.

La clase utiliza un `set` para almacenar los departamentos, garantizando que cada departamento sea único.

4.2.3. Reporte.h

```
#ifndef TRABAJOS_LP_2_REPORTE_H
#define TRABAJOS_LP_2_REPORTE_H

#include <iostream>
using namespace std;

template <typename T>
class Reportes {
public:
    static void generarReporte(T& empresa) {
        cout << "Reporte de la Empresa:" << empresa.getNombre() << endl;
        for (auto& departamento : empresa.getDepartamentos()) {
            cout << "Departamento:" << departamento->getNombre() << endl;
            for (auto& empleado : departamento->getEmpleados()) {
                cout << "  Empleado: " << empleado->getNombre() << ", Salario: " <<
empleado->calcularSalario() << endl;
            }
        }
    }

    static void mostrarSalarioTotal(const T& empresa) {
        double salarioTotal = 0;
        for (auto& departamento : empresa.getDepartamentos()) {
            for (auto& empleado : departamento->getEmpleados()) {
                salarioTotal += empleado->calcularSalario();
            }
        }
        cout << "Salario total de la empresa: " << salarioTotal << endl;
    }
};

#endif //TRABAJOS_LP_2_REPORTE_H
```

La clase `Reportes` es una plantilla que proporciona métodos estáticos para generar reportes sobre la empresa.

- **Métodos**:
- `static void generarReporte(T& empresa)`: Genera un reporte detallado de la empresa, mostrando los nombres de los departamentos y los empleados junto con sus salarios.
- `static void mostrarSalarioTotal(const T& empresa)`: Calcula y muestra el salario total de todos los empleados en la empresa.

4.2.4. Contenedor.h

```
#ifndef TRABAJOS_LP_2_CONTENEDOR_H
#define TRABAJOS_LP_2_CONTENEDOR_H
```

```
#include "Empleados/Empleado.h"
#include <vector>

// Primer módulo de empleados
Empleado* gerente = new Gerente("Alice", 5000, 2, "desarrollo", 1000);
Empleado* desarrollador = new Desarrollador("Bob", 3000, 3, "C++", 2);
Empleado* disenador = new Disenador("Charlie", 3500, 2, "Photoshop", 1);
Empleado* tester = new Tester("David", 4000, 2, "Windows", 2);

vector<Empleado*> empleados1 = {gerente, desarrollador, disenador, tester};

// Segundo módulo de empleados
Empleado* gerente2 = new Gerente("Joaquin", 5000, 2, "servicios", 3000);
Empleado* desarrollador2 = new Desarrollador("Grossman", 3000, 3, "Java", 20);
Empleado* disenador2 = new Disenador("Salvador", 3500, 2, "Illustrator", 2);
Empleado* tester2 = new Tester("Yhosfer", 4000, 2, "Linux", 3);

vector<Empleado*> empleados2 = {gerente2, desarrollador2, disenador2, tester2};

// Definición de departamentos con los empleados
Departamento<Empleado> marketing("marketing", empleados1);
Departamento<Empleado> ventas("ventas", empleados2);

#endif //TRABAJOS_LP_2_CONTENEDOR_H
```

El archivo `Contenedor.h` crea instancias de empleados y los agrupa en departamentos. Estos objetos se utilizan para inicializar el sistema y proporcionar datos de ejemplo.

- **Instancias de Empleados**:
- Crea objetos de `Gerente`, `Desarrollador`, `Disenador`, y `Tester` con nombres, salarios y otros atributos específicos.
- **Vectores de Empleados**:
- `vector<Empleado*> empleados1`: Contiene empleados del primer módulo.
- `vector<Empleado*> empleados2`: Contiene empleados del segundo módulo.
- **Departamentos**:
- `Departamento<Empleado> marketing`: Define el departamento de marketing con los empleados del primer módulo.
- `Departamento<Empleado> ventas`: Define el departamento de ventas con los empleados del segundo módulo.

4.2.5. menu.h

```
#ifndef TRABAJOS_LP_2_MENU_H
#define TRABAJOS_LP_2_MENU_H
```

```cpp
#include <iostream>
#include "Empresa.h"
#include "Contenedor.h"   // Es un archivo que contiene los datos de los empleados

class Menu {
private:
    Empresa<Departamento<Empleado>> empresa;

public:
    Menu();
    void mostrarMenuPrincipal();
    void mostrarMenuDepartamentos();
    void crearDepartamento();
    static void crearEmpleado(Departamento<Empleado>* departamento);
    void listarDepartamentos();
    static void listarEmpleados(Departamento<Empleado>* departamento);
    static void editarEmpleado(Departamento<Empleado>* departamento);
    static void eliminarEmpleado(Departamento<Empleado>* departamento);
    static void buscarEmpleado(Departamento<Empleado>* departamento);
    void generarReporte();
};

Menu::Menu() : empresa("TechCorp", &marketing) {}

void Menu::mostrarMenuPrincipal() {
    int opcion;
    do {
        cout << "1. Crear Departamento" << endl;
        cout << "2. Listar Departamentos" << endl;
        cout << "3. Generar Reporte" << endl;
        cout << "0. Salir" << endl;
        cin >> opcion;
        switch (opcion) {
            case 1:
                crearDepartamento();
                break;
            case 2:
                listarDepartamentos();
                break;
            case 3:
                generarReporte();
                break;
        }
    } while (opcion != 0);
}

// Implementación de otros métodos...
```

La clase `Menu` maneja la interfaz de usuario del sistema a través de varios métodos para interactuar con los departamentos y empleados.

- **Atributos**:
- `Empresa<Departamento<Empleado>> empresa`: Instancia de la empresa que contiene los departamentos y empleados.
- **Constructores**:
- `Menu()`: Inicializa la empresa con un nombre y un departamento de marketing predefinido.
- **Métodos**:
- `void mostrarMenuPrincipal()`: Muestra el menú principal con opciones para crear departamentos, listar departamentos y generar reportes.
- `void mostrarMenuDepartamentos()`: Muestra un submenú para gestionar los departamentos (no mostrado en el fragmento).
- `void crearDepartamento()`: Permite la creación de un nuevo departamento (no mostrado en el fragmento).
- `static void crearEmpleado(Departamento<Empleado>* departamento)`: Permite la creación de un nuevo empleado en un departamento (no mostrado en el fragmento).
- `void listarDepartamentos()`: Lista todos los departamentos existentes (no mostrado en el fragmento).
- `static void listarEmpleados(Departamento<Empleado>* departamento)`: Lista todos los empleados de un departamento específico (no mostrado en el fragmento).
- `static void editarEmpleado(Departamento<Empleado>* departamento)`: Permite editar la información de un empleado (no mostrado en el fragmento).
- `static void eliminarEmpleado(Departamento<Empleado>* departamento)`: Permite eliminar un empleado de un departamento (no mostrado en el fragmento).
- `static void buscarEmpleado(Departamento<Empleado>* departamento)`: Permite buscar un empleado en un departamento (no mostrado en el fragmento).
- `void generarReporte()`: Genera un reporte de la empresa (no mostrado en el fragmento).