Proyecto Documentación

Autor: [Tu Nombre]

Chapter 1. Introducción

Este documento describe la estructura y funcionalidad del proyecto, incluyendo explicaciones detalladas de las clases, métodos, y conceptos técnicos.

Chapter 2. Clases

2.1. Departamento

NOTE La clase `Departamento` maneja una colección de empleados y proporciona métodos para gestionar esta colección, incluyendo la adición, eliminación, edición y búsqueda de empleados. Utiliza la clase `GestorArchivos` para la persistencia de datos.

1. Atributos
   - `vector<Empleado*> empleados` - Lista de empleados (composición).
   - `GestorArchivos manejadorArchivos` - Gestor de archivos (agregación).
2. Métodos
   - `void agregarEmpleado(Empleado* empleado)` - Agrega un empleado a la lista de empleados.
   - `void eliminarEmpleado(const string& nombre)` - Elimina un empleado de la lista basado en su nombre.
   - `void editarEmpleado(const string& nombre, const string& nuevoNombre, double nuevoSalario, int nuevaFechaContratacion)` - Edita los detalles de un empleado existente.
   - `Empleado* buscarEmpleado(const string& nombre)` - Busca un empleado en la lista basado en su nombre.
   - `void listarEmpleados()` - Lista todos los empleados.
   - `void guardar(const string& archivo)` - Guarda la lista de empleados en un archivo.
   - `void cargar(const string& archivo)` - Carga la lista de empleados desde un archivo.

Explicación detallada de métodos:

`void agregarEmpleado(Empleado* empleado)`: Este método agrega un empleado a la lista `empleados` y muestra un mensaje confirmando la operación.

`void eliminarEmpleado(const string& nombre)`: Utiliza una función lambda y el algoritmo `remove_if` para eliminar un empleado basado en su nombre.

`Empleado* buscarEmpleado(const string& nombre)`: Busca un empleado utilizando `find_if` y una función lambda para comparar los nombres.

`void guardar(const string& archivo)`: Llama al método estático `guardar` de `GestorArchivos` para serializar y almacenar los datos de empleados en un archivo.

2.2. GestorArchivos

NOTE La clase `GestorArchivos` proporciona métodos estáticos para guardar y cargar

empleados desde archivos. Utiliza una fábrica de empleados para crear instancias de empleados basadas en tipos.

1. Métodos
   - `static void guardar(const string& archivo, const vector<Empleado*>& empleados)` - Guarda empleados en un archivo.
   - `static void cargar(const string& archivo, vector<Empleado*>& empleados)` - Carga empleados desde un archivo.

Explicación detallada de métodos:

`static void guardar(const string& archivo, const vector<Empleado*>& empleados)`: Este método abre un archivo y escribe la información de cada empleado, incluyendo su tipo y datos serializados.

`static void cargar(const string& archivo, vector<Empleado*>& empleados)`: Este método abre un archivo y lee la información para reconstruir las instancias de `Empleado` utilizando la fábrica de empleados y deserialización.

2.3. Empleado (Abstracto)

NOTE La clase abstracta `Empleado` sirve como base para diferentes tipos de empleados. Define los atributos comunes y métodos necesarios para todos los empleados.

1. Atributos
   - `string nombre` - Nombre del empleado.
   - `double salario` - Salario del empleado.
   - `int fechaContratacion` - Fecha de contratación del empleado.
2. Métodos
   - `void setNombre(const string& nombre)` - Establece el nombre del empleado.
   - `void setSalario(double salario)` - Establece el salario del empleado.
   - `void setFechaContratacion(int fechaContratacion)` - Establece la fecha de contratación del empleado.
   - `string getNombre() const` - Obtiene el nombre del empleado.
   - `double getSalario() const` - Obtiene el salario del empleado.
   - `int getFechaContratacion() const` - Obtiene la fecha de contratación del empleado.

Explicación detallada de métodos:

`void setNombre(const string& nombre)`: Establece el nombre del empleado.

`void setSalario(double salario)`: Establece el salario del empleado.

`void setFechaContratacion(int fechaContratacion)`: Establece la fecha de contratación del empleado.

`string getNombre() const`: Obtiene el nombre del empleado.

`double getSalario() const`: Obtiene el salario del empleado.

`int getFechaContratacion() const`: Obtiene la fecha de contratación del empleado.

2.4. Gerente

NOTE La clase `Gerente` hereda de `Empleado` y representa un tipo específico de empleado con características adicionales.

2.5. Desarrollador

NOTE La clase `Desarrollador` hereda de `Empleado` y representa un tipo específico de empleado con características adicionales.

2.6. Disenador

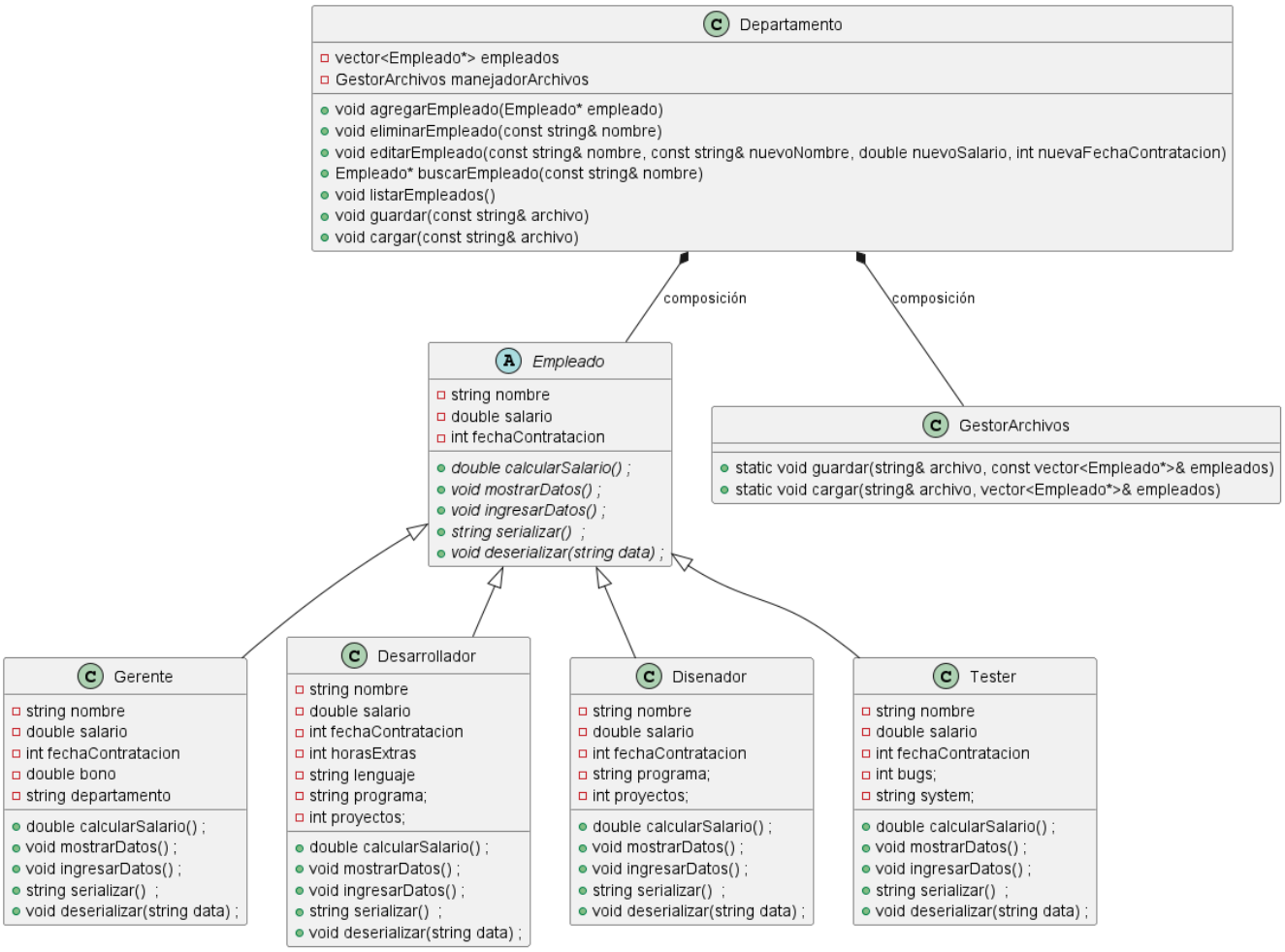
NOTE La clase `Disenador` hereda de `Empleado` y representa un tipo específico de empleado con características adicionales.

2.7. Tester

NOTE La clase `Tester` hereda de `Empleado` y representa un tipo específico de empleado con características adicionales.

Chapter 3. Diagramas

3.1. Diagrama de Clases



Chapter 4. Glosario

4.1. auto

El especificador `auto` se utiliza en C++ para la deducción automática de tipos de variables a partir de sus inicializadores. Permite al compilador determinar el tipo de una variable en función del valor con el que se inicializa.

4.2. vector<T*>

`vector<T*>` es una plantilla de clase en la biblioteca estándar de C++ que representa un contenedor dinámico que puede almacenar punteros a objetos de tipo `T`. Permite la gestión automática de la memoria y proporciona operaciones eficientes para agregar, eliminar y acceder a elementos.

4.3. function<bool(T*, T*)>

`function<bool(T*, T*)>` es una plantilla de clase en la biblioteca estándar de C++ que representa un objeto funcional, es decir, una función o un objeto que se puede llamar como una función. En este caso, representa una función que toma dos punteros a objetos de tipo `T` y devuelve un valor booleano.

4.4. Serialización y Deserialización

La serialización es el proceso de convertir un objeto en una secuencia de bytes que se puede almacenar o transmitir. La deserialización es el proceso inverso, en el que se reconstruye el objeto original a partir de la secuencia de bytes. Estos procesos son esenciales para almacenar objetos complejos en archivos o transmitirlos a través de una red.