Trabajo Practico 1

I102 – Paradigmas de Programación – G:2

Fecha límite de entrega: 16/04/2025 – 22hs

La entrega se realizará mediante un archivo comprimido ZIP que incluirá un informe y el código en C++ (.CPP y .HPP o .H). El informe, en formato PDF, debe explicar la solución de los problemas, los Warnings indicados por el compilador e incluir los comandos necesarios para compilar el código. En los archivos de C++, se deberán detallar todos los comentarios necesarios para seguir fácilmente la implementación provista. Las penalidades por no poder compilar el código y las advertencias del compilador (Warnings) no explicadas en el informe implicarán las penalidades mencionadas en el programa de la materia (50% por no compilar y entre 5-10% por Warning). El archivo de formato ZIP debe poseer el siguiente nombre TP1_NombreCompleto.zip. Recuerde que el trabajo es individual.

1. Se requiere crear el sistema de una empresa que permite la administración de sus centrales regionales, empresas, departamentos y empleados. A partir del diagrama UML de clases en la siguiente figura, escriba el código correspondiente.

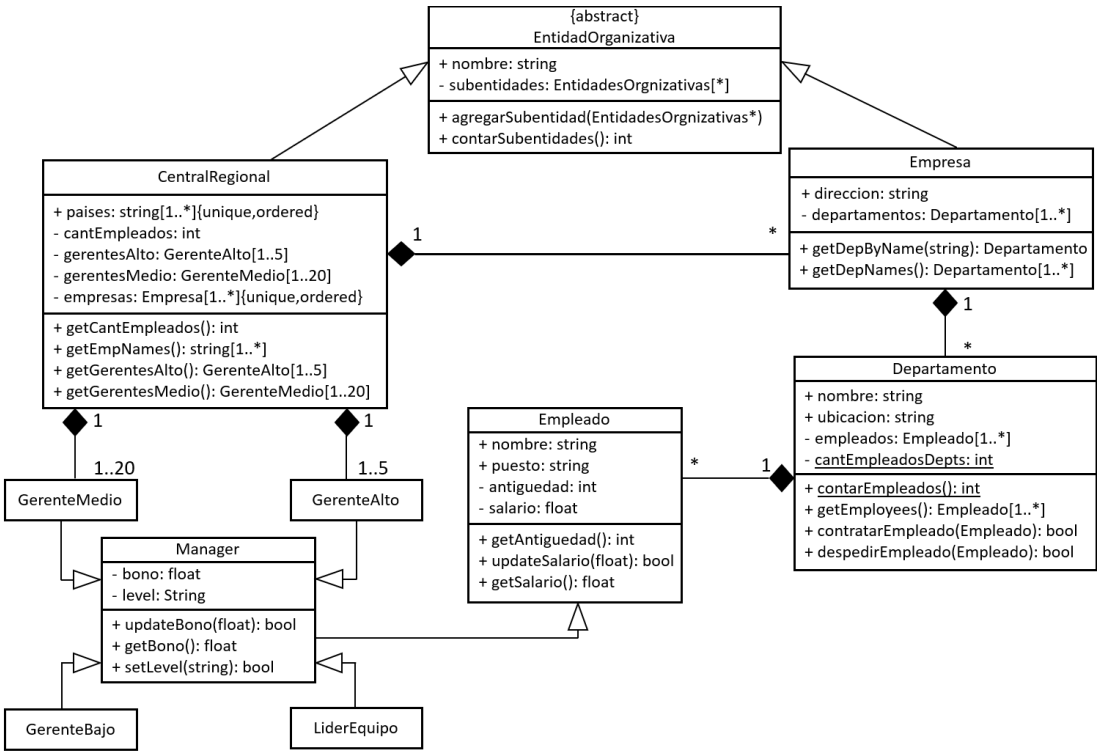

Figura 1: Diagrama de Clases UML para el ejercicio 1.

Implemente el código necesario que verifique las relaciones establecidas entre las clases según el diagrama de la Figura 1.

i. Genere varios empleados de distinto tipo.
ii. Tenga en cuenta los límites de los GerenteAlto y los GerenteMedio, y que cantEmpleadosDeps cuenta los empleados de todos los departamentos.
iii. Genere los departamentos, empresas y las centrales regionales que crea necesario.
iv. Relacione los objetos creados en los puntos anteriores de acuerdo a la Figura 1.

2. Juego de Rol: Los problemas 2, 3 y 4 están relacionados. No obstante, genere distintos proyectos o folders para cada problema. Se debe obtener un código compilado distinto en cada caso.

2.1. Interesa implementar dos grupos de armas:

i. Items Mágicos: bastón, libro de hechizos, poción y amuleto.
ii. Armas de Combate: hacha simple, hacha doble, espada, lanza y garrote.

Estas armas derivan de una interfaz única, de la cual se desprenden dos clases abstractas. Sea creativo, las clases derivadas deberán tener al menos 5 atributos y 5 métodos.

2.2. Al mismo tiempo, se desea implementar dos grupos de personajes

i. Magos: hechicero, conjurador, brujo y nigromante.
ii. Guerreros: bárbaro, paladín, caballero, mercenario y gladiador.

Al igual que en el punto anterior, estos personajes derivan de una interfaz única, de la cual se desprenden dos clases abstractas. Sea creativo, las clases derivadas deberán tener al menos 5 atributos y 5 métodos.

2.3. Considerando que cualquier personaje puede utilizar cualquier arma, realice el diagrama UML de clases de esta parte del código.

3. En un escenario del juego de rol del ejercicio 2, se desea generar en forma automática una cantidad determinada de personajes con armas. Para lograr esto, deberá:

3.1. Investigar el funcionamiento del generador de números aleatorios *std::rand()* (vea las librerías *cstdlib* y *ctime*). Con este, genere dos números enteros aleatorios. El primer número en el rango de 3-7 y el otro de 0-2.
3.2. Con el punto anterior, obtenga dos valores entre 3 y 7 para indicar la cantidad de personajes tipo Mago y la cantidad de personajes tipo Guerrero. Luego, para cada uno de estos personajes, obtenga un nuevo valor aleatorio entre 0 y 2 que indique cuantas armas tendrá cada uno de estos personajes.
3.3. Teniendo en cuenta que un personaje “has-a” arma, es decir existe una composición entre ellos. Escriba una clase denominada “PersonajeFactory” que:

i. Permita crear en forma dinámica (en run-time) los objetos tipo personajes.
ii. Permita crear en forma dinámica (en run-time) los objetos tipo armas.

iii. Permita crear en forma dinámica (en run-time) los objetos tipo personajes armados (es decir, personajes que portan armas).

Se prefiere que estas clases no tengan que ser instanciadas para retornar los objetos solicitados y que en lo posible utilicen smart pointers.
La idea es que esta clase deberá de crear un objeto, digamos un caballero, al recibir un parámetro que indique que se quiere este tipo de objeto. Luego, basándose en conceptos de polimorfismo, esta función devolverá un puntero al objeto requerido.

4. En este ejercicio se programarán la batalla entre sólo dos personajes del ejercicio 2. Las reglas de las peleas siguen un modelo de piedra-papel-tijera, en este caso “Golpe Fuerte”, “Golpe Rápido” y “Defensa y Golpe”.

4.1. Elección de ataque: El jugador 1 debe elegir por teclado una de las tres posibles opciones. La opción del jugador 2 viene dada por una variable aleatoria utilizando std::rand().

4.2. Daño y defensa: El “Golpe Fuerte” le gana al “Golpe Rápido” y hace 10 puntos de daño a quien lanzó el “Golpe Rápido”. El “Golpe Rápido” le gana a la “Defensa y Golpe” y hace 10 puntos de daño a quien lanzó “Defensa y Golpe”. Si el personaje usa “Defensa y Golpe” bloquea el “Golpe Fuerte” haciendo 10 puntos de daño a quien lanzó el “Golpe Fuerte”. En caso de que los dos personajes realicen la misma acción, ningún personaje recibirá daño y se pasa a la siguiente ronda de elección.

4.3. Interacción: Dado un personaje y un arma, el programa debe indicar el tipo de personaje y con qué arma atacó al otro personaje. Pierde el primer jugador que llega perder todos sus 100 puntos de vida (Health Points o HP). OPCIONAL: si lo desea, añada puntos dependiendo del arma utilizada.

4.4. Para iniciar a jugar, elija un personaje y un arma (el arma no modificará la pelea a no ser que implemente los puntos extra de daño), y deje que el otro personaje y arma sean elegidos aleatoriamente. Para seguir la batalla, implemente un menú sencillo, por ejemplo:

El jugador 1 tiene XXX HP y el jugador 2 tiene YYY HP.
Su opción: (1) Golpe Fuerte, (2) Golpe Rápido, (3) Defensa y Golpe: 1
El Caballero ataca con la Espada y hace 10 puntos de daño.

El Caballero tiene XXX HP y el Brujo tiene YYY-10 HP.
Su opción: (1) Golpe Fuerte, (2) Golpe Rápido, (3) Defensa y Golpe: _