Nota: Aunque el trabajo práctico incluye una carpeta correspondiente al Ejercicio 1, este fue eliminado del enunciado final luego de varias modificaciones por parte del profesor. Se decidió conservarlo en la entrega para mantener la trazabilidad del desarrollo realizado y justificar la estructura del proyecto.

El profesor indicó via Gmail:

"Si lo hubiera dejado tendrías que hacer los ejercicios 2, 3 y 4 de todas formas. Si querés entregalo y si veo que te falta puntaje para aprobar veo de tomarlo de ese ejercicio, pero los otros ejercicios de una forma u otra los tenías que hacer."

Por lo tanto, lo incluyo en mi entrega del TP1 como material adicional.

Para compilar cualquier ejercicio, por favor consulte el archivo README.md, donde se detalla el comando necesario para compilar y ejecutar cada uno.

Ejercicio 1:

Se siguió la estructura indicada por el diagrama UML provisto originalmente, pero debido a la ambigüedad y falta de especificación del diagrama, fue necesario agregar múltiples métodos adicionales para cubrir casos de uso que no estaban contemplados claramente en el enunciado como métodos para agregar y remover managers y/o países de la central regional. Se utilizaron shared_ptr en lugar de unique_ptr para permitir que distintos departamentos y empresas compartan empleados y/o managers.

Ejercicio 2 (ahora 1):

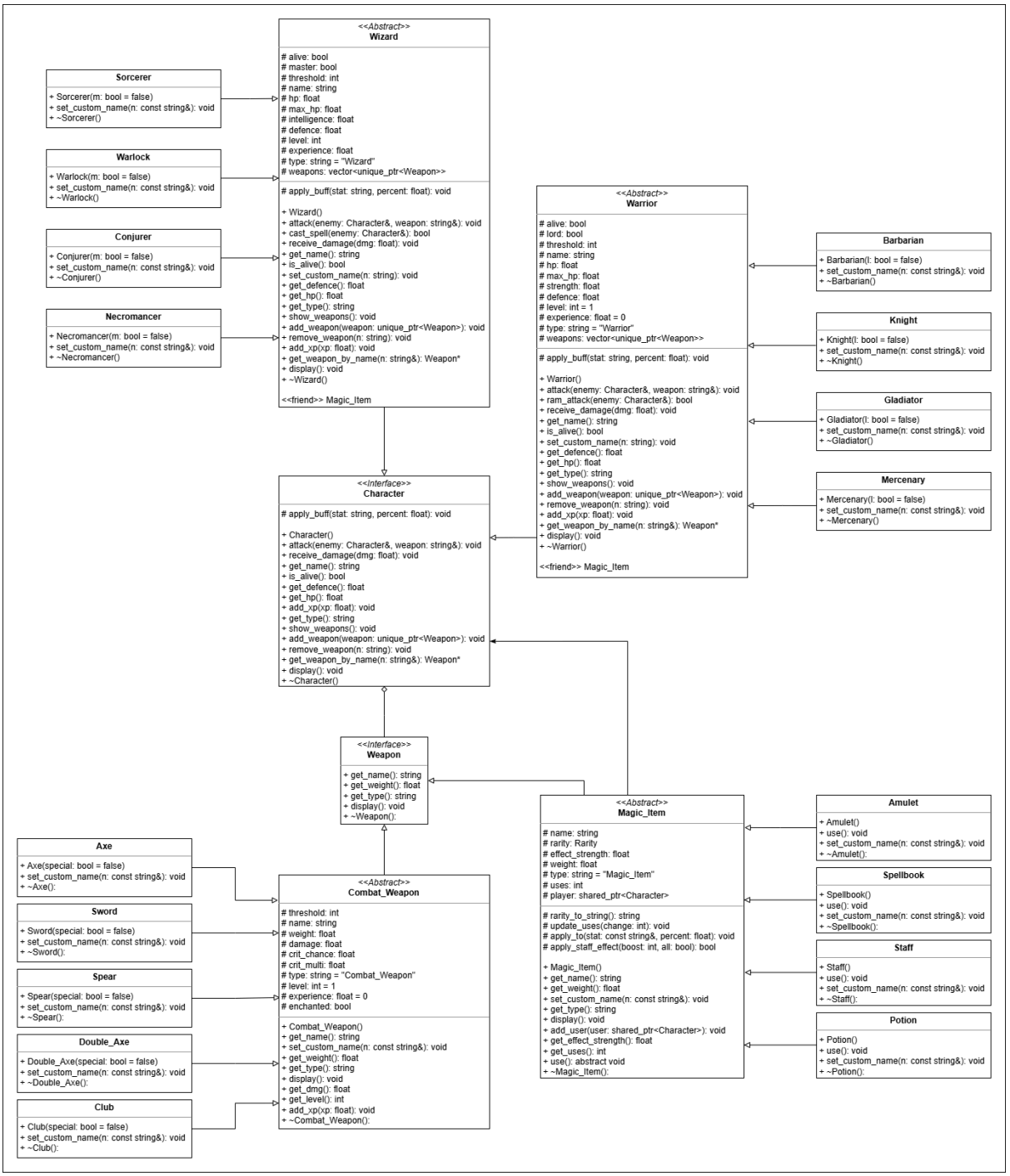
Para las armas:

- Se definió una interfaz Weapon con métodos virtuales puros.
- De esta derivan dos clases abstractas: Combat_Weapon y Magic_Item.
- Cada clase derivada (Axe, Sword, Staff, Potion, etc.) contiene al menos cinco atributos y cinco métodos, superando los mínimos requeridos.

Para los personajes:

- Se creó una interfaz Character de la cual derivan Warrior y Wizard, ambas abstractas.
- A su vez, se implementaron subclases concretas como Barbarian, Knight, Conjurer, Warlock, etc., todas con múltiples atributos y métodos propios.
- Se utilizó unique_ptr<Weapon> en los personajes para reflejar la relación de composición entre personajes y sus armas.
- En el caso de los ítems mágicos, se incluyó un shared_ptr<Character> para referenciar al dueño y aplicar efectos sobre él.

Aqui esta el UML de estas relaciones:



Ejercicio 3 (ahora 2):

Se creó la clase Character_Factory para crear de forma dinámica personajes y armas. Esta clase agrupa métodos estáticos que retornan smart pointers (unique_ptr) a objetos del tipo deseado, con posibilidad de instanciación aleatoria o específica según el método.

Para los casos aleatorios:

- Se utilizó rand() y srand(time(nullptr)), inicializado una única vez en main(), para asegurar que las instancias sean random entre ejecuciones.

- Se eligió un personaje aleatorio (de entre magos y guerreros) y un arma correspondiente a través de un número entero generado en un rango controlado.

Una vez elegido o seleccionado los personajes que se hayan creado con el Character_Factory, utilizamos el método ".display()" para mostrar los resultados.

Ejercicio 4 (ahora 3):

Se desarrolló el juego en función de las características indicadas en la consigna, implementando una batalla estilo "piedra, papel o tijera" donde cada ataque exitoso causa 10 puntos de daño al oponente y los empates no generan efectos. Para los ataques se mantuvieron todas las armas de combate previamente implementadas, mientras que los ítems mágicos fueron descartados por no tener aplicación en este ejercicio. Además, se simplificaron considerablemente los atributos de los personajes (Character), ya que no eran utilizados dentro de la dinámica de combate y resultaban innecesarios para esta instancia.