Trabajo Práctico 1
I102 – Paradigmas de Programación – G:1

¿Cómo compilar y correr los ejercicios?

Todos los ejercicios cuentan con un archivo makefile, esto facilita lo que sería tener que compilar archivo por archivo. Lo único que se debe de hacer es escribir por terminal los siguientes comandos siguiendo paso a paso:

1. Escribir “cd ejercicioN” (siendo N el número de ejercicio) para ingresar en la carpeta a compilar.
2. Escribir el comando “make”.
3. Correr el ejecutable, que en el caso del ejercicio 2 y 3 con escribir “./main” corre el código. Pero en el caso del ejercicio 1 se utiliza “./bin/main” porque el main está dentro de la carpeta bin.
4. Para finalizar, se puede ir entrando en las carpetas de cada ejercicio con “cd ejercicioN”, como se dijo antes y saliendo con “cd ..”. También si se desea eliminar los ejecutables después de haber corrido el programa compilado, se escribe el comando “make clean”.

Ejercicios:

ejercicio 1:
El ejercicio 1 consiste en crear dos interfaces (armas, personajes), de estas dos se derivan cuatro clases abstractas (**Armas**: items mágicos, de combate; **Personajes**: magos, guerreros), que heredan e implementan sus métodos.
Luego de cada una de las cuatro clases derivadas, se expanden otra cantidad de clases, que son específicas. El ejercicio pide implementar una cierta cantidad de métodos para después más adelante, poder utilizar creando un personaje con arma y poder combatir.
Cada una de las clases específicas tienen al menos algo que las hace característica entre sus hermanas (otras clases derivadas de la misma clase madre).

Estructura y lógica general del código:

1. Interfaces:

- Armas: Define las operaciones comunes para todos los tipos de armas (por ejemplo, usar(), recargar(), daño(), etc.).
- Personajes: Define las operaciones comunes para todos los personajes (por ejemplo, atacar(), defender(), cambiarArma(), etc.).

2. Clases Abstractas:

- ArmasCombate: Derivada de Armas, incluye implementaciones parciales para las armas de combate como hachas, espadas, garrotes, etc.
- ItemsMagicos: Derivada de Armas, maneja los objetos mágicos como pociones, bastones, libros de hechizos, etc.
- Magos: Derivada de Personajes, maneja el comportamiento de los magos (Hechicero, Brujo, etc.).
- Guerreros: Derivada de Personajes, maneja el comportamiento de los guerreros (Paladín, Caballero, etc.).

3. Clases Derivadas Específicas:

- Armas:
    - HachaSimple, HachaDoble, Espada, Lanza, Garrote (derivan de ArmasCombate).
    - Baston, LibroDeHechizos, Pocion, Amuleto (derivan de ItemsMagicos).
- Personajes:
    - Hechicero, Conjurador, Brujo, Nigromante (derivan de Magos).
    - Barbaro, Paladin, Caballero, Mercenario, Gladiador (derivan de Guerreros).

4. Relaciones:

- Armas y Personajes están conectadas a través de la relación de composición. Los personajes pueden tener varias armas, y cada arma tiene una relación con un personaje.
- El polimorfismo permite que los personajes usen diferentes tipos de armas sin importar su clase específica (un Hechicero puede usar tanto un bastón como una poción, y un Guerrero puede usar una espada o una lanza).

resumen:

El ejercicio te permite implementar conceptos clave de orientación a objetos, como herencia, polimorfismo, composición y el uso de interfaces.

ejercicio 2:

El ejercicio 2 tiene como objetivo generar de manera automática personajes de dos tipos: Magos y Guerreros, asignándoles un número aleatorio de armas (de 0 a 2). La creación de los personajes y las armas se realiza mediante la clase PersonajeFactory, que permite crear objetos dinámicamente en tiempo de ejecución, utilizando smart pointers para asegurar la correcta gestión de memoria.

Estructura y lógica general del código:

1. Generación de Números Aleatorios:
    - Se utiliza std::rand() para generar números aleatorios que determinan:
        - La cantidad de personajes de tipo Mago y Guerrero.
        - El número de armas (0, 1 o 2) que tendrá cada personaje.

2. Clase PersonajeFactory:
    - Esta clase tiene métodos estáticos que crean personajes y armas de manera dinámica.
    - Se usan smart pointers (shared_ptr, unique_ptr) para gestionar la memoria de los objetos creados.

3. Lógica de Asignación de Armas:
    - Para cada personaje, se elige aleatoriamente cuántas armas tendrá y se asignan armas de combate o mágicas usando PersonajeFactory::crearArma().
    - La relación entre personajes y armas es de composición (un personaje "tiene" armas).

resumen:

El ejercicio implementa un sistema eficiente para generar personajes con armas. También agregar que para manejar la memoria eficientemente y no tener memory leaks, se utilizaron smart pointers.

ejercicio 3:
El ejercicio 3 consiste en implementar una batalla entre dos personajes, una es el usuario que crea el personaje por terminal, y el otro es la máquina que se selecciona el personaje de manera random. Los personajes son creados con las funciones implementadas en el ejercicio 2, como el personajeFactory() y randomNum().
La batalla contiene un sistema de ataque basado en el clásico juego piedra, papel o tijera. En este caso las opciones son: Golpe Fuerte, Golpe Rápido y Defensa y Golpe.

Estructura y lógica general del código:

1. Generación de Ataques:
    - El jugador 1 elige su ataque por teclado, mientras que el jugador 2 lo hace aleatoriamente utilizando std::rand().
2. Interacción de Ataques:
    - Los ataques tienen una relación de daño y defensa que determina el resultado de cada ronda:
        - Golpe Fuerte vence al Golpe Rápido y causa 10 puntos de daño.
        - Golpe Rápido vence a Defensa y Golpe.
        - Defensa y Golpe vence al Golpe Fuerte.
3. Proceso del Combate:
    - Cada ronda muestra la cantidad de HP (puntos de vida) de ambos personajes antes de cada ataque.
    - Los personajes se enfrentan hasta que uno de ellos pierde todos sus HP.

resumen:

El ejercicio implementa un combate dinámico y aleatorio entre dos personajes utilizando un sistema simple de piedra-papel-tijera.

Problemas que enfrente:

Uno de los problemas que se me presentó fue tomar la decisión de ver que tipo de smart pointer usar para asignarle el arma al personaje. Estuve dudando mucho, porque más allá de que la consigna del 3 te da una pista de que un personaje “has a” arma, era más simple usar shared_ptr también evitando tener que redefinir el puntero. Finalmente llegué a la conclusión de usar unique_ptr ya que es

una manera apropiada de representar que ese arma que tiene el personaje es única y exclusiva de ese personaje.

Después otro problema fue que en algunos archivos no me estaba andando el “include namespace std” y me tiraba algunos warnings, pero lo solucione agregando adelante de algunas cosas el “std::”

Conclusión general:

El trabajo práctico me puso muy a prueba en cuanto a desarrollar mi creatividad, ya que la consigna te permite imaginar y crear una propia comprensión de lo pedido. Por otro lado, al principio sin tener del todo claro qué era lo que debía hacer, me fui un poco por las ramas, causando una gran confusión. A medida que fui avanzando en el trabajo práctico, ya pude darme cuenta de que es más o menos lo que se espera de cada ejercicio y pude acomodarme para poder ir al punto sin dispersarme.

En cuanto a los ejercicios a implementar, fueron importantes para empezar a entender el manejo de las clases y comprender conceptos clave de orientación a objetos como herencia, polimorfismo, composición. En resumen, este trabajo permite una fuerte comprensión de la estructura y diseño de un sistema basado en objetos, con un gran énfasis en la flexibilidad y extensibilidad del código.