

Trabajo Práctico 1

I102 – Paradigmas de Programación

Salvador Turkie

Universidad de San Andrés (UdeSA)

Año: 2025

Introducción

Este proyecto aborda la implementación de tres ejercicios relacionados con la programación orientada a objetos en C + +. Cada ejercicio presenta un problema específico que se resuelve mediante el uso de clases, herencia, polimorfismo y otros conceptos fundamentales de este paradigma. El proyecto está organizado en carpetas que agrupan los archivos de código fuente y encabezado según su funcionalidad.

Comandos para Compilar

El proyecto incluye un archivo makefile que automatiza el proceso de compilación. Los comandos disponibles son:

Para compilar y ejecutar el Ejercicio 1: **make ejercicio1**

Para compilar y ejecutar el Ejercicio 2: **make ejercicio2**

Para compilar y ejecutar el Ejercicio 3: **make ejercicio3**

Para limpiar los archivos generados: **make clean**

Warnings del Compilador

Durante la compilación, se utilizaron las banderas -Wall y -Wextra para habilitar advertencias adicionales. En ninguna compilación hubo WARNINGS.

Descripción de la Solución

Ejercicio 1: Diseño de Personajes y Armas

Se implementan diferentes tipos de personajes (magos y guerreros) y armas (armas de combate e ítems mágicos).

Se utiliza herencia para modelar las relaciones entre las clases base y derivadas.

El archivo main_ej1.cpp tiene como propósito:

- Validar que los personajes y armas se crean correctamente.
- Verificar que las armas se asignan adecuadamente a los personajes.
- Mostrar información detallada de los personajes y sus armas para confirmar que los métodos de las clases funcionan como se espera.

Clases principales y Derivadas

- IArma: Interfaz única, de la cual se desprenden dos clases abstractas de armas.
- IPersonaje: Interfaz única, de la cual se desprenden dos clases abstractas de personajes
- ItemsMagicos y ArmasCombate: Clases abstractas que derivan de IArma.
- Magos y Guerreros: Clases abstractas que derivan de IPersonaje.

Derivadas

- Items Mágicos: Bastón, LibroHechizos, Poción, Amuleto
- Armas de Combate: Hacha, HachaDoble, Espada, Lanza, Garrote
- Magos: Hechicero, Conjurador, Brujo, Nigromante
- Guerreros: Bárbaro, Paladín, Caballero, Mercenario, Gladiador

Diseño de clases

Considerando que cualquier personaje puede utilizar cualquier arma, se realizó el diagrama UML de clases de esta parte del código. (Figura 1)

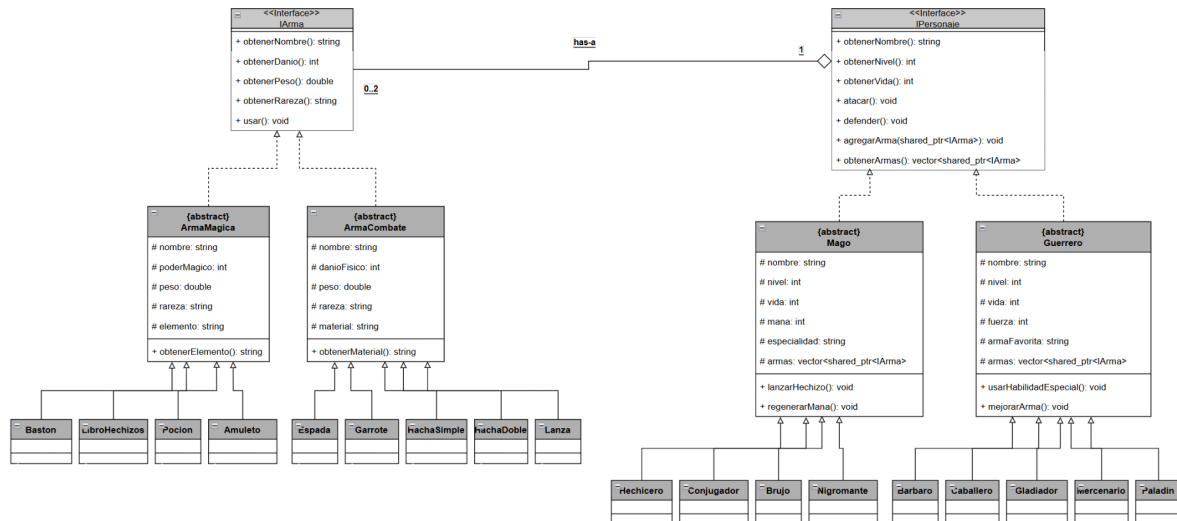


Figura 1: Diagrama UML del sistema

Aclaración: Un personaje teniendo en cuenta las condiciones del ejercicio 2, puede tener cero, una o hasta 2 armas en su vector de armas. Un arma es creada para ser usada por un personaje.

Ejercicio 2:

El ejercicio 2 se centra en la generación automática de personajes y armas.

Para generar números aleatorios, se utilizó la función `std::rand()` de la biblioteca `<cstdlib>`, inicializada con `std::srand(std::time(nullptr))` para garantizar que los números generados sean diferentes en cada ejecución. Esto se implementó en la clase **PersonajeFactory**

El método `generarNumeroAleatorio` genera un número aleatorio en un rango dado (min a max), con esto, se cumplen los requisitos de generar dos números aleatorios:

- Uno en el rango de 3 a 7 para determinar la cantidad de personajes.
- Otro en el rango de 0 a 2 para determinar la cantidad de armas.

Se implementó el método `generarPersonajesConArmas` en la clase `PersonajeFactory` para crear personajes y asignarles armas de manera aleatoria:

Generación de personajes:

Se generan dos números aleatorios entre 3 y 7 para determinar la cantidad de personajes tipo Mago y Guerrero.

Se utiliza una lista de tipos de personajes (Hechicero, Conjurador, Brujo, etc.) para seleccionar aleatoriamente el tipo de personaje.

Asignación de armas:

Para cada personaje, se genera un número aleatorio entre 0 y 2 para determinar cuántas armas tendrá.

Las armas se seleccionan aleatoriamente de una lista de tipos de armas (Espada, Garrote, Bastón, etc.).

La clase `PersonajeFactory` implementa el patrón de diseño `Factory` para crear personajes, armas y personajes armados de manera dinámica. Esto se logra mediante métodos estáticos que devuelven punteros inteligentes (`std::shared_ptr`) a los objetos creados.

Ejercicio 3:

Se implementa una simulación de batalla entre personajes utilizando las clases previamente desarrolladas, siguiendo las reglas de un modelo de piedra-papel-tijera.

El usuario tiene la libertad de elegir cualquier personaje y cualquier arma (cada arma provoca un daño adicional distinto, por lo que equivale a un nivel de dificultad diferente dependiendo de la elección), mientras que el arma y el personaje del contrincante es elegido aleatoriamente. Luego, el usuario puede seleccionar su acción, ya sea:

- “Golpe Fuerte”
- “Golpe Rápido”
- “Defensa y Golpe”

Las reglas son las siguientes:

- 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.

Para seguir la batalla, se implementa un menú claro: (ejemplo)

```
=====
¡Arranca la batalla!
Jugador 1: Barbaro
Jugador 2: Conjurador
=====

-----
Estado actual:
Jugador 1 (Barbaro): 100 HP
Jugador 2 (Conjurador): 100 HP
-----

Jugador 1, elige tu ataque:
1: Golpe Fuerte
2: Golpe Rápido
3: Defensa y Golpe
Tu elección: 3

Acciones:
Jugador 1 (Barbaro) ataca con Poción usando Defensa y Golpe.
Jugador 2 (Conjurador) ataca con Libro de Hechizos usando Defensa y Golpe.

Resultado: Hubo un empate, nadie recibió daño.
```

Fin del ejemplo

Cada personaje tiene 100 puntos de vida, y hace al menos 10 de daño, la batalla termina cuando un jugador es derrotado ($HP \leq 0$)