



# Trabajo Práctico 1

*Pilar Otero*

[potero@udesa.edu.ar](mailto:potero@udesa.edu.ar)

*2do Año Ingeniería en Inteligencia Artificial - Grupo 1 - Departamento de Ingeniería,*

*Universidad de San Andrés*

---

## **RESUMEN**

En este informe, me encargaré de explicar los diferentes aspectos del trabajo, dejando plasmado su desarrollo y composición. A su vez, se encontrarán con la forma de compilación de los ejercicios y la explicación del contenido de cada una de las carpetas.

## DESARROLLO

### 1. EJERCICIO 1 – Desarrollo de Clases

En este ejercicio se encuentra la implementación de las clases que se utilizarán durante todo el trabajo<sup>1</sup>. Así, nos encontramos con:

#### Interfaces

- Utils: métodos virtuales puros correspondientes a las armas (tanto a los Items Mágicos, como a las Armas de Combate).
- Personaje: métodos virtuales puros que serán sobrescritos por las clases derivadas (Magos y Guerreros).

#### Clases abstractas

- ItemsMagicos y ArmasCombate: clases abstractas, derivadas de Utils.
- Mago y Guerrero: clases abstractas, derivadas de Personaje.

#### Clases derivadas

- Amuleto, Bastón, Poción, LibroDeHechizos: derivadas de ItemsMagicos
- Espada, Garrote, HachaSimple, HachaDoble, Garrote: derivadas de ArmasCombate.

#### Métodos

##### 1. Interfaz Utils:

- Getters: `getNombre()`, `getDaño()`, `getDuracion()`
- Métodos: `usar()`, `mostrarInfo()`, `evolucionar(int,int)`, `involucionar(int,int)`

---

<sup>1</sup> El UML con la representación de las relaciones se encuentra dentro del repositorio.

## 2. Interfaz Personaje:

- **Getters:** `getNombre()`, `getNivel()`, `getVida()`, `getFuerza()`, `getArmas()`
- **Métodos:** `morir()`, `evolucionar(int,int)`, `mostrarInfo()`

## 3. Clase ItemsMagicos:

- **Getters:** `getPoder()`, `getMagiaActiva()`
- **Métodos:** `romper()`

## 4. Clase ArmasCombate:

- **Getters:** `getPeso()`
- **Métodos:** `reparar(int,int)`

## 5. Clase Mago:

- **Metodos:** `usarPoder()`, `revivir(int)`, `agregarVida(int)`, `involucionar(int,int)`

## 6. Clase Guerrero:

- **Getters:** `getHabilidad()`
- **Metodos:** `usarHabilidad()`, `cambiarHabilidad(string)`

Asimismo, dentro de la mayoría de las clases derivadas de las abstractas (tanto de guerreros y magos, como de ítems mágicos y armas de combate) implementé métodos adicionales.

## 2. EJERCICIO 2 – Factoría de Personajes

En este ejercicio, se desarrolló la clase `PersonajeFactory` que se encuentra compuesta por 3 métodos estáticos: `crearPersonaje(Personaje)`, el cual crea un personaje sin armas, `crearArmas(Arma)`, que genera un arma y `crearPersonajeArmado(Personaje, pair<shared_ptr<Util>, shared_ptr<Util>)`, la cual devuelve un personaje que tal como se nombre indica, cuenta con 2 armas.

Para lograr esto, es fundamental la función `generar_numero()`, que produce un número random.

Gracias a esta, se crean aleatoriamente un arma a través de `generarArmaRandom(bool)`, función a la que se le pasa como parámetro un booleano que indica si el personaje es guerrero o no; y un personaje mediante `generarPersonajeRandom(bool, pair<shared_ptr<Util>, shared_ptr<Util>)`, que produce una figura (guerrero o mago) con 2 armas (de allí 2do parámetro).

A su vez, se definen aleatoriamente la cantidad de guerreros y magos a generar (mediante la función mencionada) y se crea un vector para cada uno de los casos, al que se le agregan las nuevas identidades junto con sus armas.

### 3. EJERCICIO 3 – *Batalla*

Aquí, se desarrolla el “piedra, papel o tijera” entre un personaje elegido por el usuario, quien no sólo selecciona esto, sino también el arma y en cada ronda, la estrategia de ataque<sup>2</sup>; y un rival cuyos atributos se generan aleatoriamente.

El juego comienza con ambos jugadores teniendo 100 de HP (Health Points) y finaliza cuando alguno llega a 0. Así, en cada ronda, según el ataque de los jugadores, puede existir un empate (ambos eligen la misma estrategia) o una victoria por alguno de ellos, considerando que Golpe Fuerte le gana a Golpe Rápido, Golpe rápido le gana a Defensa y Golpe y Defensa y Golpe le gana a Golpe Fuerte.

Veamos un ejemplo de lo que se observa por consola en una ronda:

```
===== RONDA 1 =====  
Su opción: (1) Golpe Fuerte, (2) Golpe Rápido, (3) Defensa y  
Golpe  
Elija un ataque (1-3): 2  
El Conjurador tiene 100 HP y el Caballero tiene 100 HP
```

---

<sup>2</sup> TIPOS DE ATAQUE: 1) Golpe Fuerte, 2) Golpe Rápido, 3) Defensa y Golpe

```
Ambos han elegido Golpe Rápido
;Empate!
Siguiendo turno...
-----
```

## ARCHIVOS

### - Punto 1

1. `itemsYarmas` (carpeta): cada una de las clases derivadas de `ItemsMagicos` y `ArmasCombate` tiene un archivo `.cpp` y un archivo tipo header con la definición de sus métodos y atributos.
2. `magosYguerreros` (carpeta): archivos `.cpp` y headers de cada una de las clases derivadas de `Magos` y `Guerreros`.

Ambas carpetas también contienen un archivo `functions` donde se encuentran los tests a probar en el archivo `main`.

3. `main.cpp` (archivo): función principal donde se inicializan las armas y personajes y se hará la prueba de los métodos desarrollados.

### - Punto 2

1. `functions`: generación de las armas y personajes de manera random y muestra de la información por consola. En mi trabajo, opté por que los personajes mágicos sólo tengan ítems mágicos y los guerreros solo armas de combate.
2. `main`: prueba de las funciones desarrolladas.
3. `personajeFactory`: creación de los personajes tanto con, como sin armas, y generación de las armas.

### - Punto 3

1. `batalla`: desarrollo de las funciones para construir la batalla entre el jugador y el rival.

2. main: simulación de la batalla.

## COMPILACIÓN

Para compilar los archivos “main.cpp” de cada ejercicio, se deberá escribir en la terminal lo siguiente (en función del ejercicio que se desea ver):

*PUNTO 1 - acceder a la carpeta “punto 1”:*

```
g++ \  
-Wall \  
-std=c++11 \  
-g \  
-I itemsYarmas/ \  
-I magosYguerreros/ \  
main.cpp \  
itemsYarmas/*.cpp \  
magosYguerreros/*.cpp \  
-o main
```

Luego, escribir: ./main

*PUNTO 2 – acceder a la carpeta “punto 2”:*

```
g++ \  
-Wall \  
-std=c++11 \  
-g \  
../punto\ 1/itemsYarmas/*.cpp \  
../punto\ 1/magosYguerreros/*.cpp \  
functions.cpp \  
personajeFactory.cpp \  
main.cpp \  
-I ../punto\ 1/itemsYarmas \
```

```
-I ../punto\ 1/magosYguerreros \  
-o main
```

Luego, escribir: `./main`

*PUNTO 3 - acceder a la carpeta “punto 3”:*

```
g++ \  
-Wall \  
-std=c++11 \  
-g \  
../punto\ 1/itemsYarmas/*.cpp \  
../punto\ 1/magosYguerreros/*.cpp \  
../punto\ 2/functions.cpp \  
../punto\ 2/personajeFactory.cpp \  
batalla.cpp \  
main.cpp \  
-I ../punto\ 1/itemsYarmas \  
-I ../punto\ 1/magosYguerreros \  
-o main
```

Luego, escribir: `./main`

### *Observaciones*

1. `-Wall` se encarga de mostrar los Warnings, que, en este caso, son nulos, por lo que el archivo correrá perfectamente sin mostrar ninguno.
2. Pegar y copiar el código de compilación ya que de otra manera (sin agregar espacios) no funcionará la compilación del archivo.