Tecnología de Videojuegos
Práctica 4: Excepciones, E/S y colecciones

UAH, Departamento de Automática, ATC-SOL
`http://atc1.aut.uah.es`

Semana de laboratorio 4

Objetivos:

- Practicar la E/S elemental con Java
- Practicar el manejo de excepciones en Java
- Manejar a nivel básico las listas en Java
- Desarrollar buenos hábitos de programación
- Afianzar la comprensión de POO

Comentario inicial

En esta práctica se ampliará la práctica anterior añadiendo nuevas funciones. En programación, siempre hay que partir de programas sencillos y funcionales para ir añadiendo complejidad poco a poco. Ésta será la filosofía que se siga.

Ejercicio 1

Modifique el constructor de la clase `Personaje` para que pregunte al usuario el nombre del personaje.

Ejercicio 2

Modifique la clase `Personaje` para que cada personaje pueda tener un inventario de armas. Defina un nuevo atributo de la siguiente manera:

```
private List<Arma>inventario = new LinkedList<Arma>();
```

Junto con los siguientes métodos:

1. `imprimirInventario(Arma arma)`: Imprime por pantalla el inventario de armas. Para ahorrar código haga uso del método `toString()` iterando sobre los elementos de la lista `inventario`.
2. `tomarArma(Arma arma)`: Añade un nuevo arma al inventario.

3. `seleccionarArma(String arma)`: Selecciona un arma, para ello tendrá que buscar dentro del inventario el arma con el nombre dado. El arma seleccionada es el contenido del campo `arma`, dentro de la clase `Personaje`.
4. `soltarArma(String arma)`: Borra un arma del inventario.

Preguntas: ¿A qué clases afecta la incorporación del inventario? ¿cómo las afecta?

Ejercicio 3

Cree una nueva clase `Juego` que va a implementar la lógica principal del juego. El juego es muy sencillo: Se pregunta qué tipo de personaje quiere encarnar el jugador, y se crea el objeto correspondiente guardándolo en el atributo `jugador`. Se le asigna un arma. Posteriormente se crean tres enemigos, uno de cada tipo, y se guardan dentro de una lista en el atributo `enemigos`. Por último, se simula el combate del jugador contra los tres enemigos.

Para facilitar la modularidad, se pide implementar los siguientes métodos:

1. `nuevaPartida()`: Inicializa los campos `jugador` y `enemigos`
2. `reiniciar()`: Borra los campos `jugador` y `enemigos`. Se invoca al terminar una partida.
3. `iniciarJuego()`: Simula el combate entre el jugador y los enemigos.

El punto de entrada del juego es el método `main()` de la clase `Juego`, ahí se crea un objeto de la clase `Juego`, se muestra un menú al usuario ofreciendole empezar una partida o terminar, e invoca a los métodos de la clase `Juego` necesarios.

La figura 1 muestra el diagrama de clases a implementar.

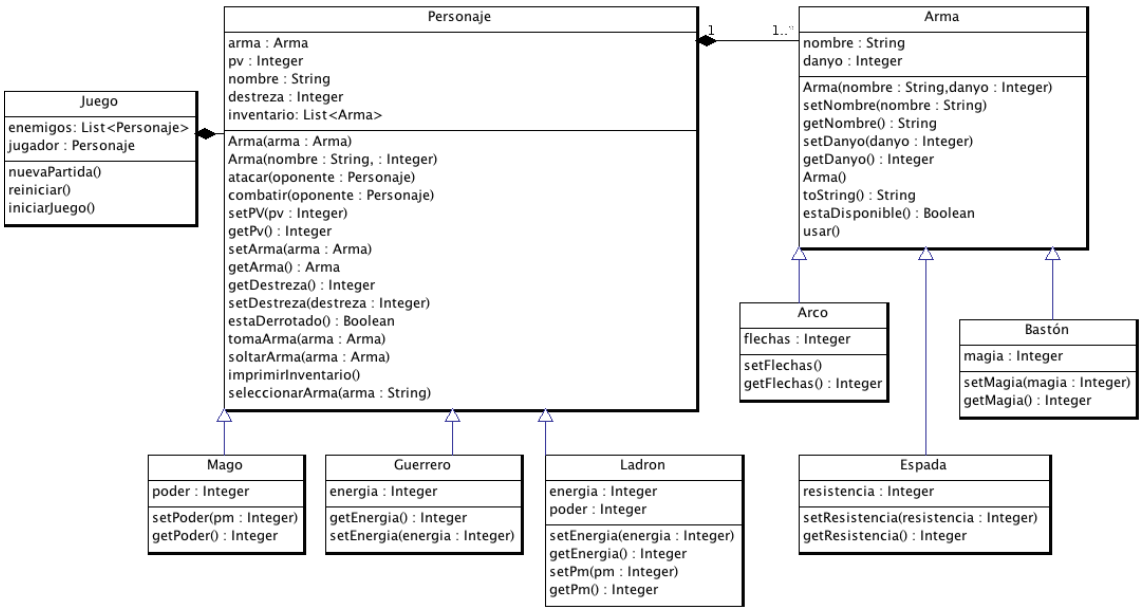

Figura 1: Diagrama de clases UML a implementar.