Tecnología de Videojuegos
Práctica 2: Herencia y sobrecarga en Java

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

Semana de laboratorio 3

Objetivos:

- Practicar el mecanismo de herencia en Java
- Practicar la sobrecarga de métodos
- Desarrollar buenos hábitos de programación
- Comprender una jerarquía de clases utilizando un diagrama de clases en UML
- Practicar los modificadores de acceso en Java
- Practicar la utilización de paquetes
- Afianzar la comprensión de POO

Comentario inicial

Se pretende desarrollar un videojuego RPG de fantasía medieval. El juego tendrá tres tipos de personaje (mago, guerrero o ladrón), cada uno con una serie de habilidades específicas. Así mismo, cada personaje podrá tener un arma (espada, arco o bastón). Como primer paso para el desarrollo del videojuego, se pretende crear una biblioteca de clases que modele los datos presentes en el juego e implemente la lógica de algunas acciones del mismo.

La práctica de programación recomendada es desarrollar las clases por separado, asegurando que cada una funciona correctamente de manera individual, para poco a poco ir integrándolas hasta obtener la aplicación final. Siguiendo esta lógica, realice los siguientes dos ejercicios.

Ejercicio 1

El primer paso en el desarrollo del videojuego es la programación de las armas de los personajes. Para ello se partirá de una clase `Arma` que servirá como clase base para el resto de las armas. De `Arma` derivan tres clases más: `Espada`, `Arco` y `Baston`.

La figura 1 representa el diagrama de clases que se pretende implementar. Hágalo siguiendo los siguientes pasos. Todas las armas tienen un nombre y un daño asociado, ambas guardadas, respectivamente, en los atributos `nombre` y `danyo`. Adicionalmente, cada arma guarda un dato específico a la misma, el arco contiene el número de flechas restante, la espada la resistencia de la misma, y el bastón puntos de magia.

Todos los atributos son protegidos mientras que todos los métodos son públicos. Los métodos get/set establecen o leen el valor del atributo respectivo.

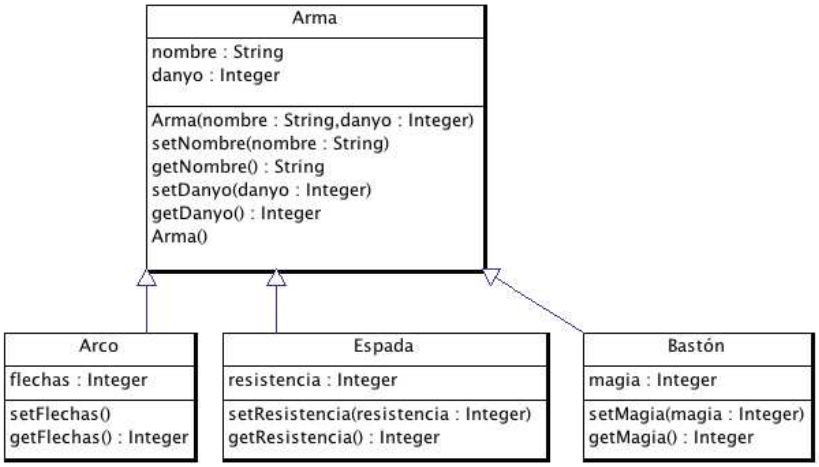

Figura 1: Diagrama de clases UML a implementar en el ejercicio 1

1. Implemente la clase `Arma` dentro del paquete `p3.armas`. Compruebe su correcto funcionamiento instanciando la clase varias veces e ejecutando los métodos dentro de `main()`.

2. Implemente las clases `Espada`, `Arco` y `Baston` dentro del paquete `p3.armas`. Compruebe su correcto funcionamiento instanciando las clases varias veces e ejecutando los métodos dentro de `main()`.

3. Implemente los constructores apropiados de las tres clases anteriores. Suponga que por defecto el arco tiene 10 flechas, la espada 20 puntos de resistencia y el bastón 15 puntos de magia. Sugerencia: Utilice un constructor con una llamada explícita al construtor de la superclase por medio de la palabra reservada `super`.

4. Sobrecargue el método `toString()` en las cuatro clases anteriores de modo que, al ser invocado, devuelva una cadena describiendo la clase en cuestión con todos sus campos. Compruebe su correcto funcionamiento.

5. Cree un vector `Arma` conteniendo objetos de tipo `Espada`, `Arco` y `Baston`. Imprima, dentro de un bucle, la descripción de todos los objetos invocando `toString()` y verifique cómo funciona el polimorfismo.

6. Se quiere implementar un método `usar()`, presente en todas las clases, que imite la utización del arma. Su comportamiento cambia en función de la clase: En la clase `Arco`, debe reducir en uno el número de flechas, en `Espada` reduce la resistencia mientras que en `Baston` decrementa la magia. Piense qué habría que cambiar en la jerarquía de clases y discutalo con el profesor antes de implementarlo.

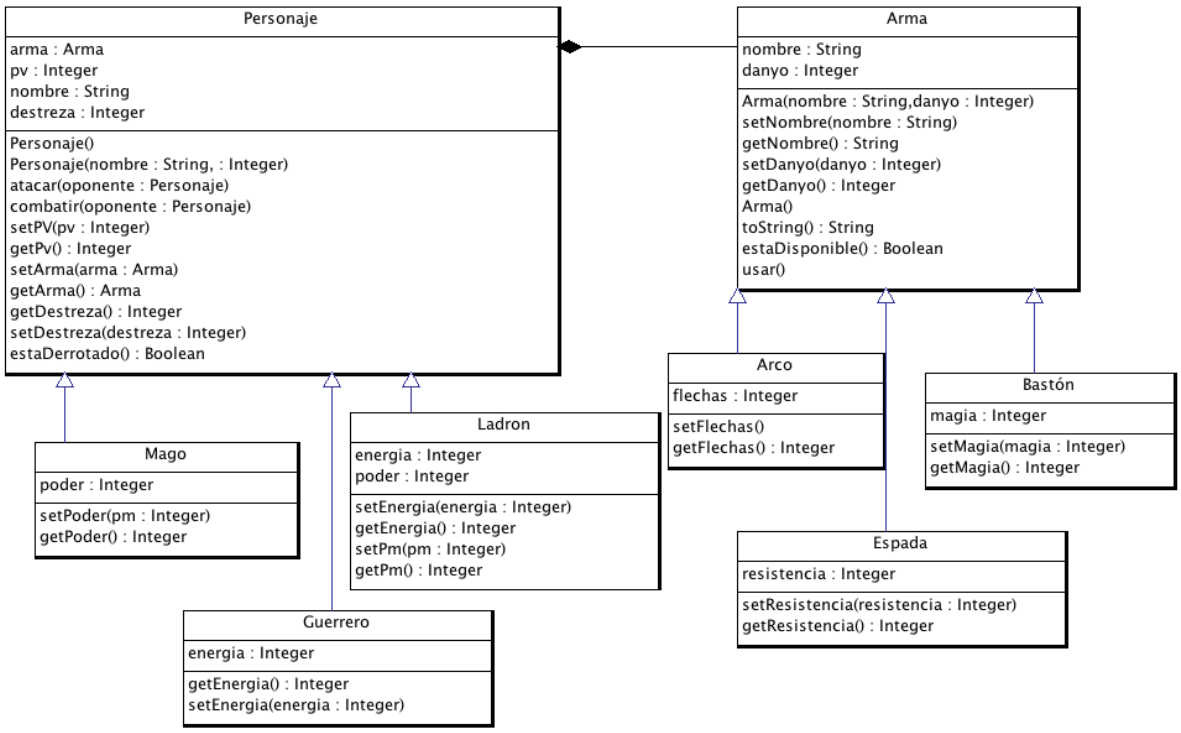

Figura 2: Diagrama de clases UML a implementar en el ejercicio 2

7. Implemente en método `estaDisponible()`, presente en todas las clases, que devuelve verdadero si el arma está disponible para el combate o falso en caso contrario. El arma esta disponible si el arco tiene flechas, la espada tiene puntos de resistencia, o el bastón tiene puntos de magia. Realice los cambios oportunos en el codigo.

Ejercicio 2

Una vez que se han implementado las clases con las armas del juego, se pretende programar las clases que representan a los personajes. Pueden ser tres: Mago, guerrero y ladrón. Cada personaje tendrá un arma y podrá luchar con otros personajes.

La figura 2 muestra el diagrama de clases completo a implementar. Hágalo siguiendo los siguientes pasos.

1. Implemente la clase `Personaje` dentro del paquete `p3.personajes`, a excepción de los métodos `atacar()` y `luchar()`. El método `estaDerrotado()` devuelve `true` si el personaje tiene cero o menos PV.

2. Implemente las clases `Mago`, `Guerrero` y `Ladron`, a excepción de los métodos `atacar()` y `luchar()`. Por defecto, el mago tendrá un bastón, el ladrón un arco y el guerrero una espada, téngalo en cuenta a la hora de programar el constructor. Sugerencia: Utilice una llamada al constructor de la superclase por medio de `super`.

3. Sobrecargue el método `toString()` en las cuatro clases anteriores para poder visualizar el contenido de los atributos fácilmente.

4. El mecanismo de combate es el siguiente. Un combate es una sucesión de ataques, primero ataca quien inicia el combate, después contrataataca el oponente, y así sucesivamente. El combate comienza invocando al método `combatir()`, cada combatiente ataca por turnos hasta que uno se queda sin PV, en cuyo caso se encuentra derrotado. El ataque se realiza invocando al método *privado* `atacar()`; al invocarse este método se restará a los PV de cada personaje el daño del arma del personaje contrario más su destreza. Después de cada ataque el mago pierde un punto de poder, el guerrero un punto de energía y el ladrón un punto de energía o poder. Si se agota cualquiera de esos elementos la destreza no suma al daño ocasionado por el personaje. Análogamente, si el arma no está disponible (arco sin flechas, espada dañada o bastón sin magia), no se suma al daño el efecto del arma. En cada ataque se invocará el método `usar()` del arma para que actualice su estado. Implemente los métodos `combatir()` y `atacar()`.