POO – CLASES

1. Continuando con el ejercicio "**Blade of Darnkness**" del tema anterior debes realizar lo siguiente:

   - Las clases Monstruo y Jugador son muy similares. Sería interesante utilizar herencia para quitar parte del código repetido. Crea una nueva clase **Personaje** que sea padre de las otras dos y lleva a ella todas las propiedades y métodos comunes. Crea esa clase como **abstraca**. Los métodos subirNivel y reducirVida son iguales, por tanto, se llevan a la clase padre. El método golpear es diferente, así que lo llevamos a la clase padre Personaje, como abstracto y cada una de las clases hijas tendrá una implementación diferente. (Recordad que las propiedades en Personaje deben ser protected).

   - Vamos a crear una nueva clase **MonstruoFinalNivel**, que herede de monstruo, añadirá como propiedades:
     - **golpeEspecial**: serán los puntos de daño de un golpe especial que tiene este monstruo.
     - **usosGolpeEspecial**: serán las veces que se ha usado el golpe especial. Se iniciará con tres, y decrementará cada vez que se use el golpe especial. Esta propiedad debe ser privada.
     - Debéis sobreescribir el método golpear para que use el golpe especial.

   - Se añade una clase nueva **Partida**. Sus atributos serán un jugador, un ArrayList de 10 (en principio) monstruos, 2 de los cuales serán MonstruoFinalNivel. Al constructor se le pasará un Jugador y, además, inicializará el ArrayList de Monstruos. Tendrá los siguientes métodos:
     - **iniciarPartida**(): se llamará en el constructor. Le añadirá en un for 100 objetos Monstruos iguales. Equipará al Jugador con las armas que decidas.
     - **turnoJugador**(): recorrerá el ArrayList de Monstruos, y el primero que encuentre le golpeará con sus armas. Si el enemigo tiene 0 o menos de vida se quitará del ArrayList de la partida. Si no quedan enemigos vivos en el ArrayList, el jugador gana la partida, y termina (return).
     - **turnoEnemigos**(): el primer Monstruo vivo que haya en el ArrayList, golpea al jugador. Si el Jugador se queda sin vida, pierde la partida y el juego acaba (return).

   - **TestJuego**. Crea un main, en él un objeto Partida, inicia la partida, y luego simula unos 30 turnos (for) alternativos de jugador y enemigo. Por último, muestra el resultado de quien gana, y cuántos enemigos destruye en total el jugador. Decide tú los valores de salud de cada uno, los puntos y daños del arma, de manera que esté ajustado el juego y el jugador pueda matar al menos 8 monstruos.

2. Equipos ciclistas

En una carrera ciclística, un **equipo** está conformado por un conjunto de ciclistas y se identifica por el nombre del equipo (tipo String), la suma de los tiempos de carrera de sus ciclistas en minutos (atributo estático) y país del equipo. Sus atributos deben ser privados.

Un **ciclista** es una clase abstracta que se describe con varios atributos: identificador (de tipo int), nombre del ciclista y tiempo acumulado de carrera (en minutos, con valor inicial cero). Los atributos deben ser privados. Un ciclista tiene un método abstracto imprimirTipo que devuelve un String.

Los ciclistas se clasifican de acuerdo con su especialidad (sus atributos deben ser privados y sus métodos protegidos). Estas especialidades no son clases abstractas y heredan los siguientes aspectos de la clase Ciclista:

- *Velocista*: tiene nuevos atributos como potencia promedio (en vatios) y velocidad promedio en sprint (Km/h) (ambos de tipo double).
- *Escalador*: tiene nuevos atributos como aceleración promedio en subida (m/s2) y grado de rampa soportada (grados) (ambos de tipo float).
- Contrarrelojista: tiene un nuevo atributo, velocidad máxima (km/h).

Definir clases y métodos para el ciclista y sus clases hijas para realizar las siguientes acciones:

- Constructores para cada clase (deben llamar a los constructores de la clase padre en las clases donde se requiera).
- Métodos get y set para cada atributo de cada clase.
- Imprimir los datos de un ciclista. Debe invocar el método de la clase padre e imprimir los valores de los atributos propios.

La clase Equipo debe tener los siguientes métodos protegidos:

- Métodos get y set para cada atributo de la clase.
- Imprimir los datos del equipo en pantalla.
- Añadir un ciclista a un equipo.
- Calcular el total de tiempos de los ciclistas del equipo (suma de los tiempos de carrera de sus ciclistas, su atributo estático).
- Listar los nombres de todos los ciclistas que conforman el equipo.
- Dado un identificador de un ciclista por teclado, es necesario imprimir en pantalla los datos del ciclista. Si no existe, debe aparecer el mensaje correspondiente.

En una clase de prueba, en un método main se debe crear un equipo y agregar ciclistas de diferentes tipos. Hacer un menú con las siguienes opciones:

- 1. Imprimir datos del equipo.
- 2. Añadir un ciclista al equipo (se van pidiendo sus datos)
- 3. Calcular el total de tiempos de los ciclistas del equipo
- 4. Listar nombres de los ciclistas
- 5. Buscar ciclista (por identificador)
- 6. Salir