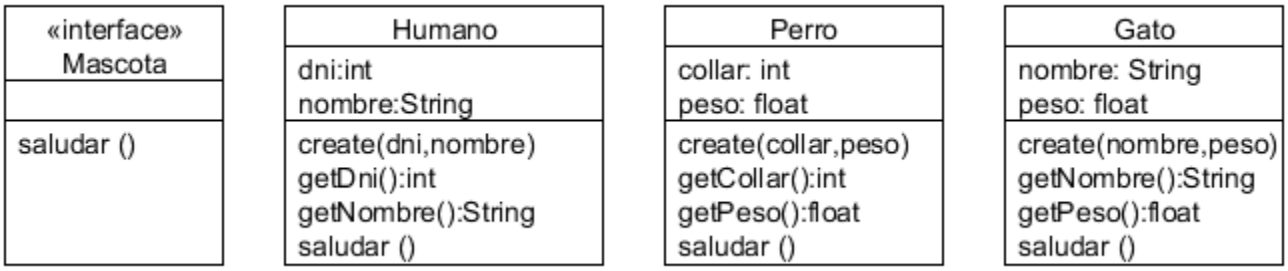
Ejercicios de colecciones de objetos (II)

En los ejercicios se trabajarán con las siguientes clases e interfaces. El significado de los métodos es evidente, excepto saludar() que muestra por pantalla “Hola”, “Guau” o “Miau”, respectivamente.



1. Crear un programa que lea los argumentos de entrada y sea capaz de crear objetos de cada una de las clases indicadas, de la siguiente forma.
   Habrá 3n argumentos de entrada, siendo “n” el número de objetos a crear. Para cada objeto indicaremos, en este orden: su clase: “H” para Humano, “P” para Perro y “G” para Gato. A continuación, dependiendo de qué tipo sea cada objeto a crear, los pares (dni,nombre) para Humano, (collar,peso) para Perro y (nombre,peso) para Gato.
   Ejemplo: H 1 Pepe H 2 Juan G micifú 4.5 P 1 30.8 G nani 8
   ...crearía 2 personas, dos gatos y un perro con la información proporcionada.
2. Ajustar **toString()** de cada clase para que devuelva algo del estilo “(H)1 Pepe”, “(P)3 35” y “(G)micifú 24.3” para humanos, perros y gatos, respectivamente.
3. Ajustar **equals()**, de manera que dos personas son iguales si coinciden en DNI, dos perros si coinciden en collar,y dos gatos si coinciden en nombre y peso.
4. Hacer que Humano, Perro y Gato implementen Comparable y el método **compareTo(...)** , de manera que un humano es siempre mayor que cualquier perro y a su vez éste mayor que cualquier gato. Los humanos se ordenan entre sí, por dni ascendente, los perros, por collar ascendente, y los gatos por nombre ascendente, y si coinciden en nombre, por peso descendente.
5. Ajustar **hashCode()** para que sea coherente con los requisitos de igualdad antes definidos.
6. Crear un **ArrayList** mezclado de humanos, perros y gatos con, al menos, un par de seres duplicados y un par de seres distintos para cada especie, cuyos datos se introduzcan mediante los argumentos de entrada.Mostrar su contenido.
7. Igual que el ejercicio anterior, pero utilizando un **LinkedList**.
8. Igual que el ejercicio anterior, pero utilizando un **HashSet**.
9. Igual que el ejercicio anterior, pero utilizando un **TreeSet**.
10. Probar a ordenar las colecciones anteriores utilizando el método **Collections.sort(...)**. ¿qué problemas encuentras?¿por qué?

11. ¿Qué ocurriría si quitáramos "**implements Comparable**" (no el método ***compareTo(...)***) de las clases Humano, Perro y Gato. Razonar y probarlo.
12. Crear una colección de Object que sólo contenga Persona's, y pedirle a todas las que contenga, que saluden.
13. Crear una colección de Object mixta de Perro's y Gato's, pedirle a todos ellos que saluden, controlando con **instanceof** hacia dónde hacer el downcasting de Object.
14. ¿Qué ocurre si se añade una Persona a la colección?
15. Hacer que Perro y Gato implementen Mascota. Hacer ahora el downcasting sin verificar a Mascota. Observar que la implementación de una interfaz común me ahorra comprobaciones.
16. ¿Qué ocurre ahora si se añade una Persona a la colección?
17. Asegurarse de que los gatos cuya letra inicial sea la "G" no saluden.
18. Hacer los ejercicios anteriores utilizando clases genéricas en las que especificamos la clase contenida y ver en qué cambia la situación para cada ejercicio.
19. Hacer un programa, que, una vez introducida una List de Humano por argumentos de entrada, pregunte por teclado por un dni y diga los datos de dicha persona, si es que existe y "No existe" en caso contrario.
20. Hacer un programa que reciba números enteros por los argumentos de entrada, y a continuación muestre la lista de números en el orden que fueron introducidos, e inmediatamente después, ordenados de menor a mayor. PISTA: Utilizar la clase wrapper Integer y el método Collections.sort(List).
21. Comprobar que JAVA ofrece sintaxis simplificada cuando trabaja con objetos wrapper en colecciones.
22. Utilizar un Map para implementar el ejercicio 19 de una forma más sencilla.