# **Ejercicio 1: Clínica veterinaria**

En esta ejercicio, implementaremos la “base de datos” de una clínica veterinaria. Se compondrá principalmente de una lista de animales, para hacer esta lista, utilizarás la clase ArrayList de Java, que nos permite trabajar con listas de objetos de una forma más sencilla que utilizando Arrays.

Como los animales tendrán diferentes características dependiendo de su especie, crearemos varias clases para las diferentes categorías que heredarán de la clase principal Animal.

Clases y Enumerados que nos hacen falta:

* Animal (clase abstracta)
  + Atributos:
    - nombre, de tipo string
    - fechaNacimiento, de tipo LocalDate
    - peso, de tipo double
    - comentarios, de tipo string
  + Métodos Get y Set (todas public para poder acceder desde fuera):
    - Se podrá leer de los cuatro atributos, pero sólo el Peso y los Comentarios serán de lectura y escritura.
  + Método abstracto:
    - toString(), para obligar a que las clases que heredan de Animal implementen obligatoriamente el toString().
* Perro (hereda de Animal)
  + Atributos adicionales:
    - raza, del tipo enumerado (ampliar tipo enumerado en Java: RazaPerro que podrá tener los valores: PastorAleman, Husky, FoxTerrier, Dalmata, SanBernardo.
    - microchip, de tipo string
  + Métodos de sólo lectura para los dos atributos adicionales.
  + Constructor al que le pasas los datos: nombre, raza, fechaNacimiento, peso, microchip (los comentarios ya se añaden luego).
  + Método toString() que sobreescribe al abstracto que servirá para mostrar por pantalla una fichita con todos los atributos de perro (incluyendo los de animal).

**Ejemplo**:

Ficha de Perro

Nombre: Luna

Raza: Pastor alemán

Fecha de Nacimiento: 12/02/2003

Peso: 12,4 kg

Microchip: 38479382749DF39 Comentarios: Está pachucho.

* Gato (hereda de Animal)
  + Atributos adicionales:
    - raza, del tipo enumerado RazaGato que podrá tener los valores: Comun, Siames, Persa, Angora, ScottishFold.
    - microchip, de tipo string
  + Constructor y Método toString igual que Perro (adaptados a los atributos de Gato).
* Pajaro (hereda de Animal)
  + Atributos adicionales:
    - especie, del tipo enumerado EspeciePajaro que podrá tener los valores: Canario, Periquito, Apaponis. No podrá modificarse una vez establecido en el constructor.
    - cantor, de tipo booleano que nos dirá si da mucho la lata por las mañanas o no.
  + Constructor y métodos igual que Perro (adaptados a los atributos de Pajaro).
* Reptil (hereda de Animal)
  + Atributos adicionales:
    - especie, del tipo enumerado EspecieReptil que podrá tener los valores: Tortuga, Iguana, DragonDeComodo. No se podrá modificar.
    - venenoso, que será de tipo boolean y nos dirá lo divertido que es que te pique un bicho de éstos.
  + Constructor y métodos igual que Perro (adaptados a los atributos de Reptil).
* ClinicaVeterinaria
  + Atributos:
    - listaAnimales será una Lista de Animal en la que meteremos todos los animales, independientemente de que sean Perros, Gatos, Pájaros o Reptiles. Utilizaremos un ArrayList.
  + Constructores: Uno básico para inicializar la lista.
  + Métodos:
    - InsertaAnimal al que le pasamos un Animal y lo añade a la lista.
    - BuscaAnimal al que le pasamos el nombre del bicho y nos devuelve los datos del Animal en cuestión.
    - ModificaComentarioAnimal al que le pasamos el nombre del animal y el nuevo comentario y nos modifica el comentario en la ficha del animal que está en la lista.
    - toString() sobrescrito, como siempre, que nos muestra la lista de todas las fichas de los animales.

Clase ejecutable:

* Crea un objeto de la clase ClinicaVeterinaria.
* Crea un menú con las siguientes opciones disponibles y haz que funcione:

1. Inserta Animal
2. Busca Animal
3. Modifica Comentario Animal
4. Muestra la lista de animales
5. Salir

**Ejercicio 2.** Dada la siguiente interface:

public interface Estadisticas  
{  
 double minimo();  
 double maximo();  
 double sumatorio();  
}

* Construir una clase **ArrayReales** que declare un atributo de tipo double[] e implemente la interface anterior. Añade a la clase ArrayReales los siguientes métodos:
  + un constructor que reciba por parámetro la dimensión del atributo array,
  + un método rellenar() que rellene el array de números reales aleatorios,
  + un método que permita añadir elementos al array uno a uno si hay espacio.
* Crea una aplicación principal que añada elementos al objeto ArrayReales y muestra después en consola el mínimo, máximo y sumatorio de dichos elementos

**Ejercicio 3.** Construye una interfaz llamada **Relaciones** que incluya los siguientes métodos:

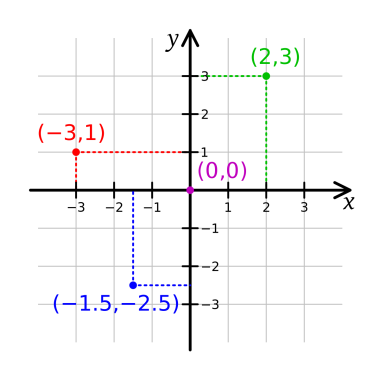
* boolean esMayor(Object b) ; // Devuelve verdadero si el objeto this es mayor que b
* boolean esMenor(Object b) ; // Devuelve verdadero si el objeto this es menor que b
* boolean esIgual(Object b) // Devuelve verdadero si el objeto this es igual que b

Crea una clase Fraccion que implemente la interface Relaciones.

La clase Fraccion tendrá como atributos un numerador y un denominador de tipo entero, varios constructores, métodos set y get y al menos métodos que permitan sumar y restar fracciones. Además de otros métodos que permitan trabajar con fracciones como simplificar …

Crea una clase principal llamada UsaFraccion que cree dos objetos Fraccion, los sume, los reste y las compare utilizando los métodos implementados de la interface Relaciones.

**Ejercicio 4.** Crea una clase principal llamada **UsaFraccion2** que cree un ArrayList de fracciones empleando la clase Fraccion del ejercicio 2. Las fracciones serán ordenadas de menor a mayor. Para ello, la clase Fraccion debe implementar la interface Comparable y redefinir el método CompareTo, el cual hará uso de los métodos de la interface Relaciones.

**Ejercicio 5.** Define la clase **Punto** que represente puntos de coordenadas cartesianas. La clase deberá contar con un constructor y métodos set/get. Además la clase deberá redefinir los métodos toString() y equals() de la clase Object y deberá permitir comparar dos Puntos. Resuelve esta última parte utilizando la interface Comparable primero y la interface Comparator después. Usa diferentes criterios de comparación para cada interface.

Crea una aplicación que cree un ArrayList de puntos y los ordene usando ambos criterios.

**Ejercicio 6.** Recupera la clase **Cafetera** (ejercicio de la relación Tarea01\_UT5\_Ejercicios).

Crea un programa principal con una colección de cafeteras (**Arraylist**) con distintas capacidades máximas y distintas cantidades actuales (puedes realizar diferentes operaciones con las cafeteras para que cambien los valores de sus atributos).

A continuación debes crear **dos comparadores** (utiliza las interfaces Comparator y Comparable) para la clase Cafetera: uno utilizará la capacidad máxima como orden natural de los elementos (interface Comparable) y otro la capacidad actual para comparar objetos (interface Comparator).

Finalmente, aplica los comparadores que has construido para ordenar la colección de cafeteras por uno u otro criterio.

**Ejercicio 7**. Define la clase **Linea** que implementa la interface Relaciones anterior. Una línea viene definida por 4 coordenadas x1, y1, x2 e y2. Implementa el método **longitud** que calcule largo de una línea en un plano de coordenadas con la fórmula de la distancia del teorema de Pitágoras:

El método toString() de la clase Linea debe devolver el detalle de ambas coordenadas y la longitud de la línea.

La clase Principal creará dos líneas y las comparará, mostrando en consola cual es la de mayor. Además se creará una colección de líneas y se ordenará usando un comparador.

**Ejercicio 8.** (Polimorfismo con Interfaces) Crea una lista de objetos Relaciones y guarda en ella objetos de tipo Linea y Fracciones. Recorre la lista e imprime el detalle de sus elementos.

**Ejercicio 9.** Añade a la interface Relaciones un método que devuelva el nombre de la clase (String) del objeto que la está utilizando de forma que no se tenga que modificar el código de las clases que implementan dicha interface. Modifica la clase Principal del ejercicio anterior para que al recorrer la lista de elementos se muestra la clase de cada objeto.

**Ejercicio 10.** Define una nueva interface de nombre Modificable con los siguientes métodos:

public interface Modificable{

void aumentar(int n);

void disminuir(int n);

default String nombreClase(){

String clase = getClass().toString();

int posicion = clase.lastIndexOf(".");

return clase.substring(posicion+1);

}

}

Modifica la clase Linea para que también implemente la interface anterior y resuelve el problema que se plantea.

**Ejercicio 11.** Modifica la interface Relaciones y añade un método estático esNull, el cual comprobará si la referencia de tipo Relaciones recibida por parámetro es null.

Crea una clase Principal con un array polimórfico de 10 posiciones y añade 5 objetos en total de diferentes tipos Línea y Fracciones. Recorre el array y muestra en consola el detalle de aquellos objetos no nulos empleando el nuevo método.