PROTECTORA DE ANIMALES

Alejandro Aranda García & Juan Rafael Escobar Díaz

FUNDAMENTOS DE PROGRAMACIÓN

ESCUELA SUPERIOR DE INFORMATICA

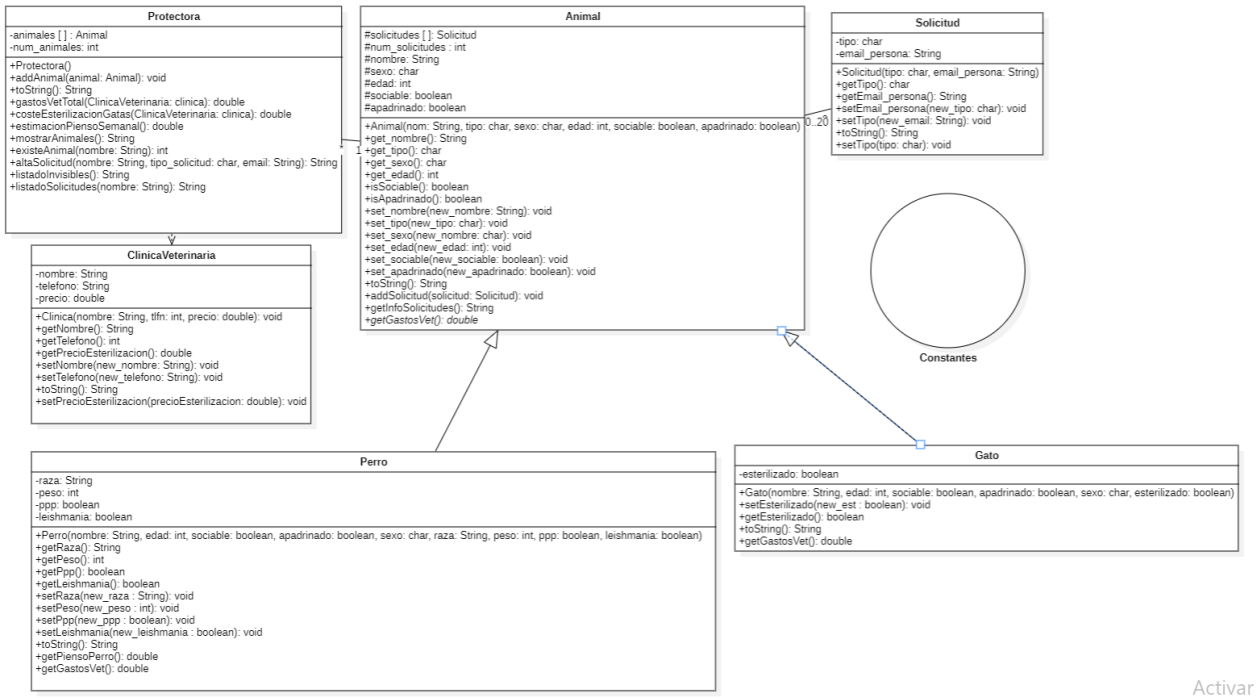
Contenido

Análisis de requisitos generales

La protectora de animales ha sido creada, en base a unos requisitos que se pedían:

- Poder saber la información de todos los animales disponibles en la protectora.
- Poder realizar una consulta de adopción o acogida a un animal
- Poder consultar las solicitudes de adopción de un animal
- Poder calcular los costes veterinarios anuales.
- Poder calcular el coste de una campaña de esterilización de gatas
- Poder calcular la estimación de la cantidad de pienso de perro adulto para una semana
- Poder consultar el listado de perros cachorros e invisibles.
- Tener un menú interactivo en el cual poder elegir que consulta de las mencionadas anteriormente elegir, incluyendo una opción para poder salir y parar de utilizar consultas

Diseño UML



Implementación

Implementación general

1. **Cargar la información de los animales desde el archivo "Animales.txt":**
   - Crearemos una clase `Animal` con atributos como nombre, tipo (perro o gato), sexo, edad, sociabilidad, etc.
   - Leeremos el archivo "Animales.txt" y crearemos objetos `Animal` para cada entrada.
   - Almacenaremos estos objetos en un array para su posterior uso.
2. **Menú de opciones:**
   - Crearemos un menú con las opciones 0, 1, 2 ,3 ,4 ,5 ,6 y 7.
   - El usuario seleccionará una opción y el sistema ejecutará la funcionalidad correspondiente.
3. **Implementación de las funcionalidades:**
   - Para cada opción, escribiremos el código necesario:
     - 0)Salir del menú.
     - 1) Mostrar toda la información de los animales
     - 2) Realizar una solicitud de adopción o acogida.
     - 3) Consultar las solicitudes de adopción de un animal.
     - 4) Calcular el coste de los gastos veterinarios anuales.
     - 5) Calcular el coste de una campaña de esterilización de gatas.
     - 6) Calcular la estimación de la cantidad de pienso de perro adulto para una semana.
     - 7) Consultar el listado de perros cachorros e "invisibles".
4. **Manejo de excepciones:**
   - Utilizaremos bloques `try-catch` para capturar excepciones como la falta del archivo "Animales.txt" o entradas no válidas.

Especificación atributos y métodos.

Clase Principal

```
import java.io.*;
import java.util.*;

public class Principal {
    final static Scanner lector = new Scanner(System.in);
    final static Scanner lectorOpcion = new Scanner(System.in);
    public static void main(String[] args) {
        boolean seguir = true;
        Protectora protectoraFranquista = new Protectora();
        try {
            leerAnimales("animales.txt", protectoraFranquista);
        } catch (FileNotFoundException e) {
            System.out.println("|-----------------------------------------
--|");
            System.out.println("ERROR AL LEER, EL FICHERO INDICADO NO
EXISTE");
            System.out.println("|-----------------------------------------
--|");
            seguir=false; // Si el fichero no existe, no continuamos con
la ejecución
        }
        if (seguir){
            ClinicaVeterniaria clinicaFranquista =
                    new ClinicaVeterniaria(14.00, "Clinica Franquista",
"6666");
            mostrarMenu();
            gestionarProtectora(protectoraFranquista, clinicaFranquista);
        }

    }
```

- Creamos la clase **Principal** donde ira contenida nuestro main. Importamos los paquetes de java que nos darán la posibilidad de usar el objeto Scanner y excepciones.
- Creamos dos objetos tipo Scanner para su posterior uso en el menú.
- Método de **leerAnimales** para poder leer del fichero los datos de todos los animales. Si no existe tal fichero se lanza una excepción y se muestra el mensaje. Además, no se seguiría con el código debido a la variable boolelana **seguir**
- Si el código sigue: se llama al método **gestionarProtectora** donde estará contenido el menú de interacción con las distintas consultas que se piden en los requisitos, se pasa como parámetro la protectora creada junto con la clínica.

```
-       public static void gestionarProtectora(Protectora
    protectoraFranquista, ClinicaVeterniaria clinicaFranquista){
-           int opcion = 2;
-           String mensaje = "";
-           String nombre;
-           String correo;
-           char tipo = 2; // Lo inicializamos en un valor diferente a
    1 y 0 para que el while se ejecute
-
-           boolean seguir = true;
-           while(opcion>0){
-               // CONTROLAR EL INPUT CON UNA EXCEPCION
-               seguir = true;
-               while (seguir) {
-                   try {
-                       opcion = askOpcion();
-                       seguir = false; // Si no se lanza ninguna
    excepción, salimos del bucle
-                   } catch (InputMismatchException e) {
-                       System.out.println("Error: Por favor, introduce
    un número entero.");
-                       lectorOpcion.nextLine(); // Limpiamos el buffer
    del scanner
-                   } catch (IllegalArgumentException e) {
-                       System.out.println("Error: Por favor, introduce
    un número entre 0 y 8");
-                   }
-               }
-
-               switch (opcion){
-                   case 0:
-                       mensaje = "Saliendo del menu..";
-                       break;
-                   case 1:
-                       mensaje =
    protectoraFranquista.mostrarAnimales();
-                       break;
-                   case 2:
-                       nombre = askNombre();
-                       tipo = askTipo();
-                       correo = askCorreo();
-
-                       try{
-                           mensaje =
    protectoraFranquista.altaSolicitud(nombre, tipo, correo);
-                       } catch (Protectora.LimiteSolicitudes e) {
-                           System.out.println("Error: " +
    e.getMessage());
-                           mensaje = "";
```

```
-                       }
-                       break;
-                   case 3:
-                       nombre = askNombre();
-                       mensaje =
    protectoraFranquista.listadoSolicitudes(nombre);
-                       break;
-                   case 4:
-                       mensaje = "Gastos previstos anuales: " +
    protectoraFranquista.gastosVetTotal() + "€";
-                       break;
-                   case 5:
-                       mensaje = "Gastos por esterilización gatas: " +
    protectoraFranquista.costeEsterilizacionGatas(clinicaFranquista) +
    "€";
-                       break;
-                   case 6:
-                       mensaje = "Gasto de pienso semanal: " +
    protectoraFranquista.estimacionPiensoSemanal() + "kg";
-                       break;
-                   case 7:
-                       mensaje =
    protectoraFranquista.listadoInvisibles();
-                       break;
-                   default:
-                       System.out.println(mensaje);
-               }
-
-               System.out.println(mensaje);
-               System.out.println();
-               mostrarMenu();
-           }
-       }
```

- Controlamos la excepción de introducir un numero valido (aquel número posible dentro del menú) con un bucle controlado con un booleano; si es verdadero se pregunta, y luego se pasa a falso para continuar el menú. Si se lanza una excepción pasa directamente a su mensaje.

```
   public static int askOpcion() throws InputMismatchException,
IllegalArgumentException{
        int opcion = -1;
        opcion = lectorOpcion.nextInt();
        if (opcion <0 || opcion > 8) {
            throw new IllegalArgumentException("La opcion debe estar
entre 0 y 8");
        }
```

```
        return opcion;
    }
}
```

- Aquí es donde lanzaremos las excepciones que hemos recogido antes.

Protectora:

Métodos

addAnimal

```
   public void addAnimal(Animal animal) {
            animales[numAnimales] = animal;
            numAnimales++;
    }
```

- Método para añadir objetos tipo **animal** al array de animales. Se usa contador para calcular la posición

costeEsterilizacion

```
    public double costeEsterilizacionGatas(ClinicaVeterniaria clinica) {
//
        double total = 0;
        for (int i = 0; i < animales.length; i++) {
            if (animales[i] instanceof Gato) {
               if (!((Gato) animales[i]).getEsterilizado()) && ((Gato)
animales[i]).getSexo()=='h') { // Si no está esterilizado, hay que pagar
el control de celo
                        total = clinica.getPrecioEsterilizacion() +
total;
                    }
                }
             }
        // Convertirmos los gastos de mensuales a anuales
        return total;
    }
```

- Método para calcular el coste de la esterilización de los gatos. Al final ponemos los gastos en anuales.

gastosVetTotal

```
    public double gastosVetTotal() {
        double total = 0;
        for (int i = 0; i < animales.length; i++) {
            if (animales[i]!=null) {
                total += animales[i].getGastosVet();   // Gastos
individuales de cada perro / gato
```

```
            }
        }
        return total; // Convertimos los gastos a gastos anuales
    }
```

- Método para calcular los gastos totales, iteramos sobre cada animal y sumamos la cantidad de sus gastos para al final devolver el gasto total.

estimacionPiensoSemanal

```
    public double estimacionPiensoSemanal(){
        double total_pienso_kg = 0;
        for (int i = 0; i < animales.length; i++) {
            if (animales[i] instanceof Perro) {
                total_pienso_kg = ((Perro) animales[i]).getPiensoPerro()
+ total_pienso_kg; // Gasto de pienso individual
            }
        }
       return total_pienso_kg*7; // Convertimos a gasto semanal
    }
```

- Método para calcular el gasto que generan los perros semanalmente, comprobamos si es un perro y obtenemos el gasto de pienso según el peso.

mostrarAnimales

```
    public String mostrarAnimales(){
        String cadena = "";
        for (int i=0;i<numAnimales;i++){
            cadena = cadena +  " "  + animales[i].toString() + "\n";
        }
        return cadena;
    }
```

- Método para mostrar la informacion de todos los animales en la consulta uno, se usa el método toString para mostrar la información de todos los animales. Se itera sobre el array de animales.

existeAnimal

```
   public int existeAnimal(String nombre){
        int existe = -1 ;  // Si no existe devuelve -1
       for (int i = 0; i < animales.length; i++) {
           if(animales[i]!=null){ // Siempre que no sea null (es decir,
exista un animal)
```

```
                if(animales[i].getNombre().toLowerCase().equals(nombre)){
// Si existe devuelve la posición del animal en el array
                    existe = i;
                }
            }
        }
        return existe;
    }
```

- Método para averiguar si ese animal existe (se lo pasamos como parámetro), se itera por todo el array animales y siempre que no sea null, existirá un animal. Pondremos un if para averiguar si dentro de esos existentes, esta el que nosotros estamos buscando, si lo encuentra marca un '1' y devolvemos la variable existe.

altaSolicitud

```
    public String altaSolicitud(String nombre, char tipo_solicitud, String
email) throws LimiteSolicitudes{
        String mensaje;
        int i_animal = existeAnimal(nombre.toLowerCase()); // Indice
animal (-1 si no existe)

        if(i_animal<0){ // Si el animal no existe
            mensaje = "No existe ningun animal con el nombre " + nombre +
" en la protectora";
        } else {

            if (animales[i_animal].getNum_solicitud() ==
NUM_MAX_SOLICITUDES) {
                throw new LimiteSolicitudes(
                        "Se ha alcanzado el numero maximo de solicitudes
(" + NUM_MAX_SOLICITUDES + ")");

            } else {
                Solicitud soli = new Solicitud(tipo_solicitud, email);
                animales[i_animal].addSolicitud(soli);
                mensaje = "Solicitud realizada correctamente";
            }
        }
        return mensaje;
    }
```

- Metodo para dar de alta una solicitud a un animal por parte del usuario. Le pasamos los parámetros de la solicitud y antes de crearla, hacemos las comprobaciones correspondientes; si existe animal y si no se ha alcanzado

el número máximo de solicitudes (lanzaremos una excepción si eso ocurre).

listadoInvisibles

```
   public String listadoInvisibles(){
        String listado = "";
       for (int i = 0; i < animales.length; i++) {
           if(animales[i]!=null){ // Siempre que no sea null (es decir, exista un animal)
               if((animales[i].getNum_solicitud()==0 && animales[i].getEdad()>5) || animales[i].getEdad()<2) { // Si existe devuelve la posición del animal en el array
                   listado = animales[i].getNombre() + " " + listado;
               }
           }
       }
       return listado;
   }
```

- Método para mostrar los animales invisibles de la protectora, es decir, los cachorros y aquellos mayores de 5 años y sin solicitudes.

listadoSolicitudes

```
   public String listadoSolicitudes(String nombre){
        String listado = "";
        int i_animal = existeAnimal(nombre);
        if(i_animal<0){
            listado = "El animal solicitado no existe";
        } else {
            listado = animales[i_animal].getInfoSolicitudes();
        }
        return listado;
   }
```

- Método para mostrar la información de cada solicitud para el animal solicitado (pedido en parámetro). Primero comprueba si existe y si es así, pide la información.

ExcepcionCreada

```
    class LimiteSolicitudes extends Exception {
        public LimiteSolicitudes (String mensaje){
            super(mensaje);
        }
    }
}
```

- Creamos una clase llamada **LimiteSolicitudes** hija de la lase Exception, para poder controlar so el array de solicitudes para ese animal ha alcanzado el límite máximo de solicitudes que se pueden crear para ese animal.

Clase ClinicaVeterinaria

Atriburos

```
    private double precioEsterilizacion;
    private String nombre;
    private String telefono;
```

- Estos son los atributos establecidos. Modificado private.

Métodos

Getters & Setters

```
public double getPrecioEsterilizacion() {
        return precioEsterilizacion;
    }
    public void setPrecioEsterilizacion(double precioEsterilizacion){
        this.precioEsterilizacion = precioEsterilizacion;
    }
    public String getNombre() {
        return nombre;
    }
    public String getTelefono() {
        return telefono;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }
```

- Métodos getters y setters para cada atributo de la clase **ClinicaVeterinaria**

toString

```
    public String toString() {
        return "ClinicaVeterniaria{" + "precioEsterilizacion=" +
precioEsterilizacion + '}';
    }
```

- Método toString.

Clase Solicitud

Atributos

```
    private char tipo;
    private String email_persona;
```

- Aitrbutos de la clase Solicitud. Modificador private.

Métodos

Getters & Setters

```
 public char getTipo() {
        return tipo;
    }

    public String getEmail_persona() {
        return email_persona;
    }

    public void setTipo(char tipo) {
        this.tipo = tipo;
    }

    public void setEmail_persona(String email_persona) {
        this.email_persona = email_persona;
    }
```

- Métodos getters y setters para cada atributo de la clase **Solicitud**

toString

```
    public String toString() {
```

```
        return "Solicitud{" + "tipo=" + tipo + ", email_persona='" +
email_persona + '\'' + '}';
    }
}
```

- Método toString de la clase **Solicitud**

Clase Perro

Obtiene herencia de la clase **Animal**.

Atributos

```
    private String raza;
    private int peso;
    private boolean ppp;
    private boolean leishmania;
```

- Atributos de la clase **Perro**. Modificadores private.

Métodos

Getters & Setters

```
    public void setRaza(String raza) {
        this.raza = raza;
    }

    public void setPeso(int peso) {
        this.peso = peso;
    }

    public void setPpp(boolean ppp) {
        this.ppp = ppp;
    }

    public void setLeishmania(boolean leishmania) {
        this.leishmania = leishmania;
    }

    public String getRaza() {
        return raza;
    }

    public int getPeso() {
        return peso;
    }
```

```
    public int getEdad() {
        return edad;
    }

    public boolean getPpp() {
        return ppp;
    }

    public boolean getLeishmania() {
        return leishmania;
    }
```

- Getters y Setters de la clase **Perro.**

getPiensoPerro

```
    public double getPiensoPerro(){
        int peso_animal;
        double pienso_kg = 0;
        if (getEdad()>2) { // Cuando los animales no son cachorros
calculamos el pienso
            peso_animal = getPeso();
            if (peso_animal<=15) { // Peso menor o igual a 15kg
                pienso_kg=PIENSO_PERROS_PEQUENOS;
            } else if (peso_animal>25) { // Peso mayor a 25kg
                pienso_kg= peso_animal*PIENSO_PERROS_GRANDES;
            }else { // Peso entre 15 y 25 kg
                pienso_kg=PIENSO_PERROS_MEDIANOS;
            }
        }
        return pienso_kg;
    }
```

- Método para calcular el pienso que va a consumir el perro en cuestión. Si la edad de ese perro no es mayor de 2 años, entonces no calcularemos su pienso. Lo contamos con un If. Obtenemos el peso del perro y si es menor o igual a 15, entonces le asignaremos la constante parra perros pequeños. Si el perro pesa entre 15 y 20 le asignaremos la cantidad para perros medianos.

getGastosVet

```
    public double getGastosVet(){
        double total = 0;
        if (!isApadrinado()) { // Si no está apadrinado..
```

```
            if (getLeishmania()) {
                total = PRECIO_LEISHMANIA*12 + total;
            }
            if (getPpp() && !isSociable()) {
                total = VACUNA_RABIA + PRECIO_SEDACION + total; // Coste
vacuna rabia (anual)
            } else {
                total +=VACUNA_RABIA; // Coste anual rabia
            }
        }
        return total;
    }
```

- Método para calcular el gasto de ese perro para la vacuna, rabia, sedación. Si ese perro no esta apadrinado, si ese perro tiene leishmania le asignaremos el precio de la leishmania, además lo multiplicaremos por 12 para obtener su gasto anual. Si es perro potencialmente peligroso y no es social, le añadiremos la vacuna de la rabia y el precio de sedación.

toString

```
    public String toString() {
        return super.toString() + " | Raza: " + raza + " | Peso " + peso
+ "(kg) | PPP: " + ppp + " | Leishmania: " + leishmania;
    }
}
```

- Método toString de la clase **Perro**

Clase Gato

La clase gato recibe herencia de la clase **Animal**.

Atributos

```
private boolean esterilizado;
```

- Atributo de la clase **Gato**. Modificador private.

Métodos

Getters & Setters

```
   public boolean getEsterilizado(){
        return esterilizado;
```

```
    }

    public void setEsterilizado(boolean esterilizado) {
        this.esterilizado = esterilizado;
    }
```

- Getter y Setter de la clase **Gato**

getGastosvet

```
   public double getGastosVet(){
        double total= 0;
        if (!isApadrinado()) { // Si no está apadrinado..
            if (!getEsterilizado() &&  getSexo()=='h') { // Si no está
esterilizado, hay que pagar el control de celo
                total = COSTE_CONTROL_CELO + total;
            }
        }
        return total*12;
    }
```

- Método para calcular el gasto que ocupa ese gato. Si el gato esta apadrinado, no se asumirá ningún coste por parte de la protectora. Si el gato NO esta apadrinado entonces sí. Si ese gato no esta esterilizado y su sexo es hombre, entonces le asignaremos la constante del coste de control del celo, al final multiplicaremos el resultado por 12 para calcular el gasto anual.

toString

```
   public String toString(){

        return super.toString() + " | Esterilizado: " + esterilizado;
    }
```

- Método toString de la clase **Gato**.

Constantes (Interfaz)

```
public interface Constantes {
    int NUM_MAX_ANIMALES = 20;
    int NUM_MAX_SOLICITUDES = 20;
    int COSTE_CONTROL_CELO = 14;
    double PRECIO_LEISHMANIA = 35.00;
    double VACUNA_RABIA=40.00;
    double PRECIO_SEDACION=20.00;
    double PIENSO_PERROS_PEQUENOS=0.2;
    double PIENSO_PERROS_MEDIANOS=0.3;
    double PIENSO_PERROS_GRANDES=0.015;
}
```

- Se define interfaz llamada **Constantes** para la creación de variables fijas que se utilizaran para el desarrollo del código. Se pondrán los valores de los precios que tiene la clinica con la que trabaja la protectora y también el número máximo de los arrays

Manual de usuario

Programa para realizar consultas a nuestra protectora. Se podrán realizar las aquellas que muestre el menú:

Ya sea que la operación haya sido correctamente realizada o no, se informará al usuario con un mensaje por pantalla.

El funcionamiento de cada opción es el siguiente:

0) Salir del menú.

1) Mostrar toda la información de los animales

2) Realizar una solicitud de adopción o acogida.

3) Consultar las solicitudes de adopción de un animal.

4) Calcular el coste de los gastos veterinarios anuales.

5) Calcular el coste de una campaña de esterilización de gatas.

6) Calcular la estimación de la cantidad de pienso de perro adulto para una semana.

7) Consultar el listado de perros cachorros e “invisibles”.

Porcentaje de aportación

Alejandro Aranda García => 60%

Juan Rafael Escobar Diaz => 40%