Universidad de Málaga

ETSI Informática

ANDAMIAJE JAVA PARA MODELO DE GESTIÓN DE REFUGIO DE ANIMALES

Artina Aradnasab

Soraya Bennai Sadqi

Luis Alberto Castaño Quero

María Victoria Huesca Peláez

Marcos Luque Montiel

Francisco Ramírez Cañadas

Juan José Serrano España

Grupo 3-5

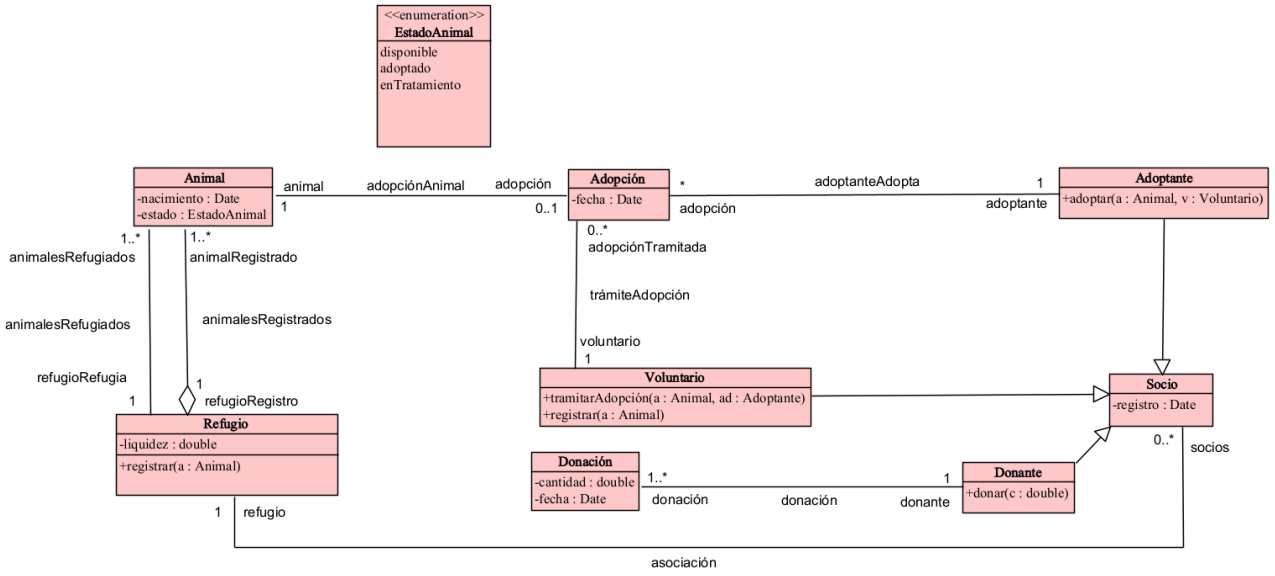
ÍNDICE

INTRODUCCIÓN

Este proyecto tiene como objetivo desarrollar un sistema de gestión para un refugio de animales. El sistema permite registrar socios, gestionar adopciones, registrar donaciones, y manejar animales y sus estados. La implementación se diseñó siguiendo principios de programación orientada a objetos (POO), garantizando extensibilidad, encapsulación y modularidad. A continuación, se detalla el diseño del sistema, la implementación de las clases, y las decisiones justificadas detrás de cada apartado.

APARTADO A

1. Diagrama de clases



En el diagrama actualizado, la clase Adopción que antes era una clase asociación se ha modificado creando una relación bidireccional entre Animal y Adopción y otra relación bidireccional entre Adopción y Adoptante.

2. Implementación en Java

2.1. SOCIO

```
import java.time.LocalDate;
```

```java
import java.util.*;

public class Socio {
    // Atributo de la clase
    private LocalDate registro;

    // Variables que representan las relaciones
    private Refugio refugio;

    // Contructor
    public Socio(LocalDate fecha, Refugio refugio){
        assert(fecha != null && refugio != null); // Nos aseguramos de
que ni la fecha ni el refugio sean nulos
        this.registro = fecha;
        this.refugio = refugio;
    }

//----------getters/setters--------------

    private LocalDate getRegistro() {
        return registro;
    }

    private void setRegistro(LocalDate registro) {
        assert(registro != null);
        this.registro = registro;
    }

    protected Refugio getRefugio(){
        return refugio;
    }

    private void setRefugio(Refugio refugio){
        assert(refugio != null);
        this.refugio = refugio;
    }
}
```

Esta es la clase base de los socios del refugio. Sirve para definir lo esencial que cualquier socio tiene que tener, como la fecha en la que se registró y el refugio al que pertenece. A partir de aquí, se crean las subclases con cosas más específicas, como donantes, adoptantes o voluntarios.

- **Atributos:**
    - LocalDate registro: Fecha en la que el socio se unió al refugio.

    - Refugio refugio: Representa el refugio con el que está relacionado el socio.
- **Métodos:**
    - getRegistro() y getRefugio(): Métodos para consultar la fecha y el refugio del socio.
    - setRegistro() y setRefugio(): Métodos que permiten modificar esos valores, asegurándose de que no sean nulos usando validaciones con assert.

2.2. DONANTE

```
import java.time.LocalDate;
import java.util.*;

public class Donante extends Socio{
    // Variables que representan las relaciones
    private List<Donacion> donaciones;

    // Constructor
    public Donante(LocalDate fecha, Refugio refugio) {
        super(fecha, refugio); // Llamamos al constructor de socio
        this.donaciones = new LinkedList<Donacion>();
        System.out.println("Donante creado.");
    }

//----------getters/setters--------------

    public List<Donacion> getDonaciones() { // Visibilidad pública para
que se le pueda llamar desde otras clases
        return donaciones;
    }

    private void setDonaciones(List<Donacion> donaciones){
        assert(donaciones != null);
        this.donaciones = donaciones;
    }

    private void agregarDonacion(Donacion donacion) {
        assert(donacion != null);
        donaciones.add(donacion);
    }

//-----------Métodos-------------
    public void donar(float c){
```

```
        assert(c > 0); // Nos aseguramos de que la cantidad de la
donación sea positiva
        Donacion donacion = new Donacion(c,LocalDate.now());
        agregarDonacion(donacion);
        Refugio ref = this.getRefugio();
        ref.setLiquidez(ref.getLiquidez()+c); // Agregamos la cantidad
a la liquidez del refugio asociado
        System.out.println("Donación realizada. Donacion: " + c);
        System.out.println("Nueva liquidez del refugio: " +
ref.getLiquidez());
    }

}
```

La clase Donante es una extensión de Socio y es para los socios que hacen donaciones al refugio. Su objetivo principal es gestionar y registrar las donaciones que realizan.

- *Atributos principales:*
    - **List<Donacion> donaciones:** Una lista donde se guardan todas las donaciones realizadas por el donante. Esto permite llevar un registro detallado.
- *Métodos destacados:*
    - **donar(float c):** Permite al donante realizar una nueva donación. Se crea un objeto Donacion, se guarda en la lista y, además, se actualiza la liquidez del refugio. Realizando validaciones para que la cantidad donada sea positiva.
    - **getDonaciones()**: Este método permite consultar la lista de donaciones hechas por el donante.

Esta clase es útil para llevar un control claro de las donaciones y para que los fondos del refugio estén siempre actualizados y manteniendo un registro de los mismos.

2.3. DONACIÓN

```
import java.time.LocalDate;

public class Donacion {
    // Atributos de la clase
    private float cantidad;
    private LocalDate fecha;

    // Constructor
```

```
    public Donacion(float cantidad, LocalDate fecha) {
        assert(fecha != null && cantidad > 0); // Nos aseguramos de que
ninguno de los parámetros sea nulo
        this.cantidad = cantidad;
        this.fecha = fecha;
        System.out.println("La donación ha tenido lugar de manera
correcta, con la cantidad "+ this.cantidad +  " y la fecha: "+
this.fecha);
    }

    //----------- getters ------------

    public float getCantidad() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        return cantidad;
    }

    private LocalDate getFecha() {
        return fecha;
    }

    //----------- setters -------------

    private void setCantidad(float cantidad) {
        assert(cantidad > 0);
        this.cantidad = cantidad;
    }

    private void setFecha(LocalDate fecha) {
        assert(fecha != null);
        this.fecha = fecha;
    }
}
```

La clase Donación modela una contribución económica realizada al refugio. Se registran dos datos necesarios: la cantidad donada y la fecha en que se realizó.

- *Atributos:*
    - **float cantidad:** Representa el monto de dinero donado.
    - **LocalDate fecha:** Indica el día exacto en que se hizo la donación.
- *Constructor:*
    - Recibe como parámetros la cantidad y la fecha de la donación.

- Valida que la cantidad sea positiva y que la fecha no sea nula usando assert.
- Imprime un mensaje de confirmación para asegurarse de que la donación se creó correctamente.

- *Métodos:*
  - **getCantidad()**: Devuelve la cantidad donada y tiene acceso público, ya que otras clases (como el refugio o donante) necesitan consultar dicho dato.
  - Métodos privados como **setCantidad()** y **setFecha()**.

2.4. VOLUNTARIO

```
import java.time.LocalDate;
import java.util.*;

public class Voluntario extends Socio {
    // Constructor
    public Voluntario(LocalDate fecha, Refugio refugio) {
        super(fecha, refugio); // Llamamos al constructor de socio
    }

    public void tramitarAdopcion(Adoptante ad, Animal a) {
        assert(ad != null && a != null); // Nos aseguramos de que
ninguno de los parámetros sea nulo
        if (a.getEstado() == EstadoAnimal.DISPONIBLE) { // Comprobamos
que el animal esté disponible para ser adoptado
            Date fecha = new Date();
            Adopcion adop = new Adopcion(fecha, a , ad, this);
            ad.getAdopciones().add(adop); // Añadimos la adopción a la
lista de adopciones del socio adoptante
            a.setAdopcion(adop);
            a.setEstado(EstadoAnimal.ADOPTADO);
            this.getRefugio().getAnimalesRefugiados().remove(a); //
Eliminamos el animal de la lista de refugiados del refugio
            System.out.println("Adopción registrada: Se ha adoptado al
animal. Estado actual del animal: " + a.getEstado());
        } else {
            System.out.println("El animal no está disponible para
adopción. Estado actual del animal: " + a.getEstado());
        }
    }
```

```
    public void registrar(Animal a) {
        this.getRefugio().registrar(a);
    }
}
```

La clase Voluntario también extiende Socio, pero su enfoque está en la gestión de las adopciones y el registro de animales en el refugio. Este rol es esencial para coordinar las interacciones entre los adoptantes y los animales disponibles.

- *Métodos destacados:*
    - **tramitarAdopcion(Adoptante ad, Animal a):** Este método gestiona el proceso de adopción. Valida que el animal esté en estado DISPONIBLE antes de crear una nueva instancia de Adopcion. Actualiza el estado del animal a ADOPTADO, lo elimina de la lista de animales refugiados del refugio y registra la adopción en la lista del adoptante. Si el animal no está disponible, informa al usuario mediante mensajes del programa.
    - **registrar(Animal a):** Este método permite a los voluntarios registrar nuevos animales en el refugio, marcándolos automáticamente como disponibles para adopción. Para ello usa el método registrar() de la clase Refugio.

Esta clase hace que los voluntarios sean el puente entre el refugio y los adoptantes, asegurando que las operaciones de adopción y registro sean coherentes y controladas.

2.5. ADOPCIÓN

```
import java.util.*;

public class Adopcion {
    // Atributo de la clase
    private Date fecha;

    // Variables que representan las relaciones
    private Animal animal;
    private Adoptante adoptante;
    Voluntario voluntario;

    // Constructor
    public Adopcion(Date fechaAdop, Animal a, Adoptante adop,
Voluntario v){
        assert(fechaAdop!=null && a!=null && adop!=null && v!=null); //
Nos aseguramos de que ninguno de los parámetros sea nulo
```

```
        this.fecha=fechaAdop;
        this.animal=a;
        this.adoptante=adop;
        this.voluntario=v;
        System.err.println("Adopcion creada.");
    }

//-------------getters---------------
    private Date getFecha(){
        return fecha;
    }

    private Animal getAnimal(){
        return animal;
    }

    private Adoptante getAdoptante(){
        return adoptante;
    }

    private Voluntario getVoluntario(){
        return voluntario;
    }
///---------------setters--------------

    private void setFecha(Date fechaAdop){
        assert(fechaAdop!=null);
        this.fecha=fechaAdop;
    }

    private void setAnimal(Animal a){
        assert(a!=null);
        this.animal=a;
    }

    private void setAdoptante(Adoptante adop){
        assert(adop!=null);
        this.adoptante=adop;
    }

    private void setVoluntario(Voluntario v){
        assert(v!=null);
        this.voluntario=v;
```

```
    }
}
```

Esta clase representa el evento de adopción en el refugio. Cada adopción asocia un animal, un adoptante y el voluntario que gestionó el proceso, además de la fecha en que ocurrió.

- *Atributos:*
    - **Date fecha:** Fecha en que se realizó la adopción.
    - **Animal animal:** Animal que fue adoptado.
    - **Adoptante adoptante:** Persona que adoptó al animal.
    - **Voluntario voluntario:** Voluntario encargado de gestionar la adopción.
- *Constructor*:
    - Recibe como parámetros todos los elementos esenciales para una adopción.
    - Valida que ninguno sea nulo con assert.
    - Imprime un mensaje de confirmación cuando la adopción se crea exitosamente.
- *Métodos*:
    - Los métodos **getFecha()**, **getAnimal()**, **getAdoptante()** y **getVoluntario()** permiten consultar los datos de la adopción, aunque están privados para mantener el encapsulamiento.

En pocas palabras, esta clase asegura que cada adopción esté completa y correctamente registrada, conectando los participantes y el animal involucrado.

2.6. ADOPTANTE

```
import java.util.List;
import java.time.LocalDate;
import java.util.*;


public class Adoptante extends Socio {
    // Variables que representan las relaciones
    private List<Adopcion>adopciones;

    // Constructor
    public Adoptante (LocalDate fecha, Refugio refugio){
        super(fecha, refugio); // Llamamos al constructor de Socio
        adopciones = new LinkedList<Adopcion>();
        System.out.println("Se ha registrado el adoptante correctamente
en el sistema, con la fecha de registro: " + fecha);
    }
```

```
    //----------getters/setters-------------

    public List<Adopcion> getAdopciones(){
        return adopciones;
    }

    private void setAdopciones(List<Adopcion> adop){
        assert(adop != null);
        this.adopciones = adop;
    }

    //---------funciones-----------------------
    public void adoptar(Animal a, Voluntario v){
        assert(a != null && v != null); // Nos aseguramos de que
ninguno de los parámetros sea nulo
         if(a.getEstado() == EstadoAnimal.DISPONIBLE) {
            v.tramitarAdopcion(this, a);
         }else{
            System.err.println("El animal no está disponible para
adopción.");
         }
    }
}
```

Esta clase modela a los socios del refugio que adoptan animales. Se extiende de Socio y añade características específicas relacionadas con las adopciones.

- *Atributos:*
    - **List<Adopcion> adopciones:** Lista donde se guardan todas las adopciones realizadas por el adoptante.
- *Constructor*:
    - Inicializa la lista de adopciones y llama al constructor de la clase Socio para establecer los datos básicos (fecha de registro y refugio).
    - Imprime un mensaje para confirmar el registro del adoptante.
- *Métodos*:
    - **adoptar(Animal a, Voluntario v):** Este es el método clave. Verifica que el animal esté disponible (DISPONIBLE) antes de delegar la gestión al voluntario. Si el animal ya está adoptado o en tratamiento, imprime un mensaje de error.
    - **getAdopciones()**: Devuelve la lista de adopciones del adoptante.

En resumen, esta clase organiza y controla todo lo relacionado con los adoptantes y sus adopciones.

2.7. ANIMAL

```
import java.time.LocalDate;

public class Animal {
    // Atributos de la clase
    private LocalDate nacimiento;
    private EstadoAnimal estado;

    // Variables que representan las relaciones
    private Adopcion adopcion;
    private Refugio refugioRegistrado;

    // Constructor
    public Animal (LocalDate fecha, Refugio refugio){
        assert(fecha != null && refugio != null); // Nos aseguramos de
que ninguno de los parámetros sea nulo
        this.nacimiento = fecha;
        this.estado = null;
        this.adopcion = null;
        this.refugioRegistrado = refugio;
        System.out.println("El animal se ha creado correctamente con la
fecha de nacimiento: " + this.nacimiento + " y se ha registrado en el
refugio.");
        this.refugioRegistrado.registrar(this); // Registramos el
animal en el refugio
    }

    //----------- getters -----------
    private LocalDate getNacimiento(){
        return nacimiento;
    }

    public EstadoAnimal getEstado(){ // Visibilidad pública para que se
le pueda llamar desde otras clases
        return estado;
    }

    private Adopcion getAdopcion(){
        return this.adopcion;
```

```
    }

    //----------- setters -----------
    private void setNacimiento(LocalDate nacimiento){
        assert(nacimiento != null);
        this.nacimiento = nacimiento;
    }

    public void setEstado(EstadoAnimal estado){
        assert(estado != null);
        this.estado = estado;
    }

    public void setAdopcion(Adopcion adopcion){
        assert(adopcion != null);
        this.adopcion = adopcion;
    }

    private Refugio getRefugioRegistrado(){
        return this.refugioRegistrado;
    }

    private void setRefugioRegistrado(Refugio refugio){
        assert(refugio != null);
        this.refugioRegistrado = refugio;
    }
}
```

Esta clase representa a los animales que se encuentran registrados en el refugio. Cada animal cuenta con información básica como su fecha de nacimiento y su estado actual, además de estar vinculado a un refugio y una posible adopción.

- *Atributos*:
    - **LocalDate nacimiento:** Fecha de nacimiento del animal.
    - **EstadoAnimal estado:** Enum que define el estado actual del animal (DISPONIBLE, ADOPTADO o ENTRATAMIENTO).
    - **Adopcion adopcion:** Asociación con una adopción, si aplica.
    - **Refugio refugioRegistrado:** Refugio donde está registrado el animal.
- *Constructor*:
    - Valida que la fecha de nacimiento y el refugio no sean nulos.
    - Registra automáticamente al animal en el refugio usando el método registrar().

- *Métodos*:
    - **setEstado(EstadoAnimal estado):** Cambia el estado del animal tras validar que sea uno de los permitidos.
    - **setAdopcion(Adopcion adopcion):** Vincula al animal con una adopción válida.

2.8. REFUGIO

```
import java.util.ArrayList;
import java.util.List;

public class Refugio{
    // Atributo de la clase
    private float liquidez;

    // Variables que representan las relaciones
    private List<Animal> animalesRegistrados;
    private List<Animal> animalesRefugiados;
    private List<Socio> sociosRefugio;

    // Constructor
    public Refugio() {
        this.liquidez = 0;
        this.animalesRegistrados = new ArrayList<>();
        this.animalesRefugiados = new ArrayList<>();
        this.sociosRefugio = new ArrayList<>();
        System.out.println("Se ha creado el refugio correctamente.");
    }

//----------getters/setters--------------

    public float getLiquidez() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        return liquidez;
    }

    public void setLiquidez(float liquidez) { // Visibilidad pública
para que se le pueda llamar desde otras clases
        assert(liquidez >= 0);
        this.liquidez = liquidez;
    }
```

```
    public List<Animal> getAnimalesRegistrados() { // Visibilidad
pública para que se le pueda llamar desde otras clases
        return animalesRegistrados;
    }

    public List<Animal> getAnimalesRefugiados() { // Visibilidad
pública para que se le pueda llamar desde otras clases
        return animalesRefugiados;
    }

    private List<Socio> getSociosRefugio() {
        return sociosRefugio;
    }

//----------métodos--------------

    public void registrar(Animal a) {
        assert(a != null);
        animalesRegistrados.add(a); // Añadimos el animal a la lista de
registrados
        animalesRefugiados.add(a); // Añadimos el animal a la lista de
refugiados
        a.setEstado(EstadoAnimal.DISPONIBLE); // Cambiamos el estado
del animal a disponible
        System.out.println("Animal registrado correctamente en el
refugio.");
    }
}
```

La clase Refugio gestiona los recursos y listas de animales y socios asociados.

- *Atributos*:
    - **float liquidez:** Representa los fondos disponibles.
    - **List<Animal> animalesRegistrados:** Lista con todos los animales que han pasado por el refugio.
    - **List<Animal> animalesRefugiados:** Lista con los animales que aún están en el refugio.
    - **List<Socio> sociosRefugio:** Lista con todos los socios asociados.
- *Constructor*:
    - Inicializa las listas y establece la liquidez en cero.
    - Imprime un mensaje confirmando que el refugio ha sido creado correctamente.
- *Métodos*:

- **registrar(Animal a):** Agrega un animal a las listas de animalesRegistrados y animalesRefugiados, y lo marca como DISPONIBLE.
- **getLiquidez()** y **setLiquidez(float liquidez):** Permiten consultar y actualizar los fondos del refugio, asegurando que no sean valores negativos.

2.9. ESTADO ANIMAL

```
public enum EstadoAnimal {
    DISPONIBLE,
    ADOPTADO,
    ENTRATAMIENTO
}
```

El enum EstadoAnimal define los posibles estados en los que puede estar un animal dentro del sistema del refugio. Los valores disponibles son:

- **DISPONIBLE:** Indica que el animal está listo para ser adoptado.
- **ADOPTADO**: Indica que el animal ya ha sido adoptado.
- **ENTRATAMIENTO**: Indica que el animal está bajo algún tipo de tratamiento, lo que puede limitar su disponibilidad para adopción.

Este enum proporciona una manera clara y sencilla de manejar los estados del animal, asegurando que solo se utilicen valores predefinidos en el sistema.

2.10. MAIN

```
import java.time.LocalDate;

public class Main {
    public static void main(String[] args) throws Exception {
        // Creo una fecha actual y un refugio
        LocalDate fecha = LocalDate.of(2023,11,28);
        LocalDate fecha2 = LocalDate.of(2024,5,14);
        LocalDate fecha3 = LocalDate.of(2022,12,25);
        Refugio refugio = new Refugio();
        System.out.println();

        //Creamos un animal y un adoptante
        Animal animal = new Animal(fecha2, refugio);
        Animal animal2 = new Animal(fecha3, refugio);
```

```
        System.out.println("Número total de animales en el refugio: " +
refugio.getAnimalesRegistrados().size() + "\n");

        Adoptante adoptante = new Adoptante(fecha, refugio);
        Adoptante adoptante2 = new Adoptante(fecha, refugio);
        Voluntario voluntario = new Voluntario(fecha, refugio);
        System.out.println("Número de animales refugiados: " +
refugio.getAnimalesRefugiados().size());
        adoptante.adoptar(animal, voluntario);
        adoptante.adoptar(animal2, voluntario);
        System.out.println("Número de adopciones del adoptante1: " +
adoptante.getAdopciones().size());
        System.out.println("Número de animales refugiados: " +
refugio.getAnimalesRefugiados().size() + "\n");

        adoptante2.adoptar(animal, voluntario);
        System.out.println("Número de adopciones del adoptante2: " +
adoptante2.getAdopciones().size() + "\n");

        Donante donante = new Donante(fecha, refugio);
        donante.donar(950);
        System.out.println();
        donante.donar(50);
        System.out.println("Número total de donaciones del donante1: "
+ donante.getDonaciones().size() + "\n");
    }
}
```

2.11. SALIDA POR PANTALLA

```
Se ha creado el refugio correctamente.

El animal se ha creado correctamente con la fecha de nacimiento: 2024-05-14 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.
El animal se ha creado correctamente con la fecha de nacimiento: 2022-12-25 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.
Número total de animales en el refugio: 2

Se ha registrado el adoptante correctamente en el sistema, con la fecha de registro: 2023-11-28
Se ha registrado el adoptante correctamente en el sistema, con la fecha de registro: 2023-11-28
Número de animales refugiados: 2
Adopcion creada.
Adopción registrada: Se ha adoptado al animal. Estado actual del animal: ADOPTADO
Adopcion creada.
Adopción registrada: Se ha adoptado al animal. Estado actual del animal: ADOPTADO
Número de adopciones del adoptante1: 2
Número de animales refugiados: 0

El animal no está disponible para adopción.
Número de adopciones del adoptante2: 0

Donante creado.
La donación ha tenido lugar de manera correcta, con la cantidad 950.0 y la fecha: 2024-12-02
Donación realizada. Donacion: 950.0
Nueva liquidez del refugio: 950.0

La donación ha tenido lugar de manera correcta, con la cantidad 50.0 y la fecha: 2024-12-02
Donación realizada. Donacion: 50.0
Nueva liquidez del refugio: 1000.0
Número total de donaciones del donante1: 2
```

El Main prueba la funcionalidad completa del sistema del refugio. Inicializa fechas, un refugio, animales, adoptantes, un voluntario y un donante, verificando la interacción entre ellos.

Se simulan adopciones mediante un voluntario, comprobando la disminución de animales refugiados y el registro de adopciones en los adoptantes. También se prueban donaciones, registrando cada aportación y actualizando la liquidez del refugio. Además, se gestionan casos como intentar adoptar un animal ya adoptado, mostrando cómo el sistema maneja la disponibilidad.

El programa valida la integración entre las clases y la actualización de datos clave como listas de adopciones, donaciones y animales refugiados.

APARTADO B

Estas clases no pueden implementarse directamente en Java ya que ésta no permite la herencia múltiple. Por ejemplo, un socio puede tener múltiples roles (Voluntario, Donante, Adoptante) que están representados como clases independientes, y un diseño basado únicamente en herencia directa no funcionaría en Java ya que eso implicaría que un socio no pueda desempeñar varios roles al mismo tiempo de manera directa.

APARTADO C

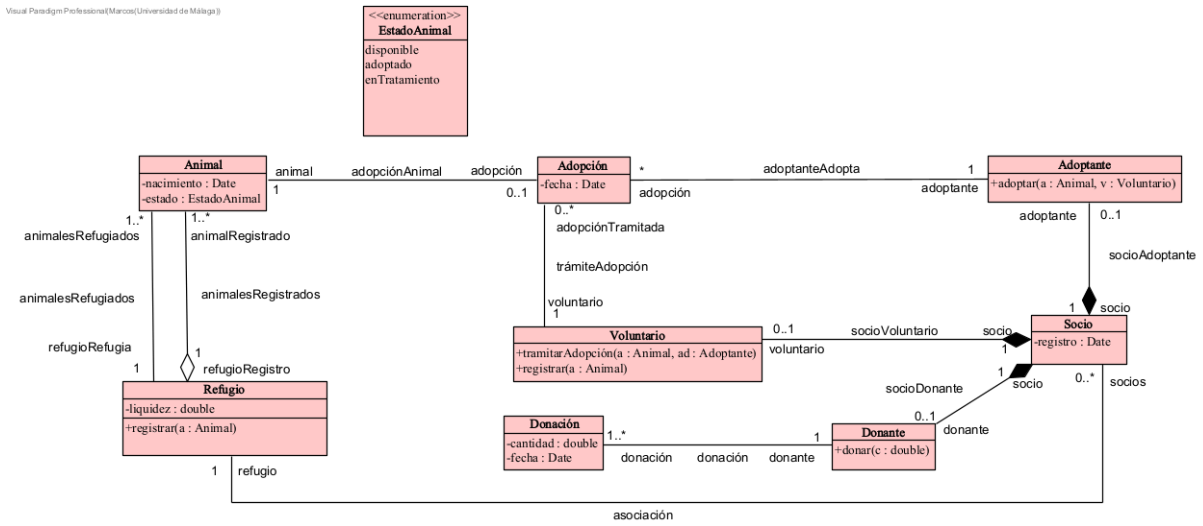
Para que un mismo socio pueda tener más de un rol, hemos optado por hacerlo mediante composiciones. En lugar de usar herencia como antes, ahora la clase Socio tiene tres variables internas que representan esos roles (Adoptante, Donante y Voluntario). Dependiendo de los parámetros booleanos (roles) que se declaren en el constructor o se asignen a lo largo de la ejecución mediante los setters, se inicializan unas variables u otras, pudiendo acceder a los métodos correspondientes de cada rol.

Si un socio activa un rol (por ejemplo, el de Adoptante), la variable correspondiente se inicializa y ya puede usar los métodos propios de ese rol. Si más adelante decide desactivarlo (es decir, el booleano se pone en false), perderá acceso a esos métodos, aunque anteriormente estuvieran habilitados. Esto permite que un mismo socio pueda cambiar de roles o incluso tener varios al mismo tiempo según lo necesite.

Este diseño es mucho más flexible que la herencia, ya que un socio puede empezar siendo solo adoptante y, más adelante, convertirse también en donante o voluntario. Además, centralizamos toda la información en una sola clase, lo que simplifica el modelo y evita duplicar datos. Así mantenemos todas las funcionalidades del sistema anterior, pero ahora con la posibilidad de que un socio tenga múltiples roles de forma dinámica, sin complicar el diseño del sistema.

APARTADO D

1. Diagrama de clases



El diagrama actualizado reemplaza la herencia entre Socio y las clases Adoptante, Donante y Voluntario, debido al uso de composición. Permitiendo de este modo que un socio tenga múltiples roles al mismo tiempo o pueda cambiar de rol. Ahora Socio incluye los atributos que representan estos roles : Adoptante, Donante y Voluntario, habilitando y deshabilitando cada rol según sea necesario. Con este cambio se eliminan las limitaciones de herencia y se mantiene intacta la funcionalidad que se pedía.

2. Implementación en Java

2.1. SOCIO

```java
import java.util.*;
import java.time.LocalDate;

public class Socio {
    // Atributo de la clase
    private LocalDate registro;
```

```java
    // Variables que representan las relaciones
    private Refugio refugio;
    private Adoptante adoptante;
    private Donante donante;
    private Voluntario voluntario;

    // Constructor
    public Socio(LocalDate fecha, Refugio refugio, boolean
rolAdoptante, boolean rolDonante, boolean rolVoluntario) {
        assert(fecha != null && refugio != null); // Nos aseguramos de
que ni la fecha ni el refugio sean nulos
        this.registro = fecha;
        this.refugio = refugio;
        // Inicializamos las variables cuyos parámetros rol estén a
true
        if(rolAdoptante) {
            this.adoptante = new Adoptante(this);
        } else {
            this.adoptante = null;
        }
        if(rolDonante) {
            this.donante = new Donante(this);
        } else {
            this.donante = null;
        }
        if(rolVoluntario) {
            this.voluntario = new Voluntario(this);
        }else {
            this.voluntario = null;
        }
    }

    //--------- getters --------------

    private LocalDate getRegistro() {
        return registro;
    }

    public Refugio getRefugio() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        return refugio;
    }
```

```
    public Adoptante getAdoptante() { // Visibilidad pública para que
se le pueda llamar desde otras clases
        assert(adoptante != null);
        return adoptante;
    }

    public Donante getDonante() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        assert(donante != null);
        return donante;
    }

    public Voluntario getVoluntario() { // Visibilidad pública para que
se le pueda llamar desde otras clases
        assert(voluntario != null);
        return voluntario;
    }


    //------------ setters -------------

    private void setRegistro(LocalDate registro) {
        assert(registro != null);
        this.registro = registro;
    }

    private void setRefugio(Refugio refugio) {
        assert(refugio != null);
        this.refugio = refugio;
    }

    public void setRolAdoptante(boolean rolAdoptante) { // Visibilidad
pública para que se le pueda llamar desde el main
        if(!rolAdoptante){
            this.adoptante = null;
        }
        if(rolAdoptante && this.adoptante == null){
            this.adoptante = new Adoptante(this);
        }
    }

    public void setRolDonante(boolean rolDonante) { // Visibilidad
pública para que se le pueda llamar desde el main
```

```
        if(!rolDonante){
            this.donante = null;
        }
        if(rolDonante && this.adoptante == null){
            this.donante = new Donante(this);
        }
    }

    public void setRolVoluntario(boolean rolVoluntario) { //
Visibilidad pública para que se le pueda llamar desde el main
        if(!rolVoluntario){
            this.voluntario = null;
        }
        if(rolVoluntario && this.voluntario == null){
            this.voluntario = new Voluntario(this);
        }
    }

    //--------------- Método --------------------------

    public String getRoles() { // Devuelve un string con los roles que
tenga el socio
        StringBuilder roles = new StringBuilder("Los roles que tiene el
socio son: ");
        boolean tieneRol = false;

        // Verificar cada rol
        if (voluntario != null) {
            roles.append("Voluntario");
            tieneRol = true;
        }
        if (donante != null) {
            if (tieneRol) roles.append(", "); // Si ya hay otro rol, se
agrega una coma
            roles.append("Donante");
            tieneRol = true;
        }
        if (adoptante != null) {
            if (tieneRol) roles.append(", ");
            roles.append("Adoptante");
            tieneRol = true;
        }
```

```
        // Si no tiene ningún rol
        if (!tieneRol) {
            roles.append("Ninguno");
        }

        return roles.toString();
    }
}
```

La nueva versión de la clase Socio reemplaza la herencia que tenía antes con Adoptante, Donante y Voluntario por un sistema basado en la composición. Ahora, un socio puede tener múltiples roles al mismo tiempo o cambiar de roles. Esto se controla con tres atributos que representan los roles y que se activan o desactivan según los parámetros booleanos que se pasen al constructor o mediante los métodos setRolAdoptante, setRolDonante y setRolVoluntario. Así, un socio puede empezar con un rol y luego ir añadiendo o quitando otros roles según sea necesario.

Además, la clase ahora centraliza todo lo relacionado con los roles, sin necesidad de depender de subclases. Por ejemplo, con el método getRoles, se puede consultar fácilmente qué roles tiene activos un socio en un momento dado. De esta manera el sistema es más práctico, más sencillo de aplicar a distintas situaciones y más fácil de adaptarse.

2.2. DONANTE

```
import java.util.*;
import java.time.LocalDate;

public class Donante {
    // Variables que representan las relaciones
    private List<Donacion> donaciones;
    private Socio socio;

    // Constructor
    public Donante(Socio socio) {
        this.donaciones = new LinkedList<Donacion>();
        this.socio = socio;
        System.err.println("Donante creado.");
    }
```

```
//----------getters/setters-------------

    public List<Donacion> getDonaciones() { // Visibilidad pública para
que se le pueda llamar desde otras clases
        return donaciones;
    }

    private void setDonaciones(List<Donacion> donaciones){
        assert(donaciones != null);
        this.donaciones = donaciones;
    }

//-----------Métodos-------------

    private void agregarDonacion(Donacion donacion) {
        assert(donacion != null);
        donaciones.add(donacion);
    }

    public void donar(float c){
        assert(c > 0); // Nos aseguramos de que la cantidad de la
donación sea positiva
        Donacion donacion = new Donacion(c,LocalDate.now());
        agregarDonacion(donacion);
        Refugio ref = socio.getRefugio();
        ref.setLiquidez(ref.getLiquidez() + c); // Agregamos la
cantidad a la liquidez del refugio asociado
        System.out.println("Donación realizada. Donacion: " + c);
        System.out.println("Nueva liquidez del refugio: " +
ref.getLiquidez());
    }
}
```

La clase Donante ahora incluye una variable socio, que establece una relación directa con un socio específico. Esto hace que el donante dependa del socio al que pertenece. En el constructor, simplemente se pasa un objeto Socio y se guarda en esta variable, eliminando la necesidad de herencia. También se ha cambiado el método donar(), que ahora usa socio.getRefugio() para acceder al refugio asociado, en lugar de depender de this. Esto hace que todo sea más claro y coherente con el nuevo enfoque de composición.

2.3. DONACIÓN

```
import java.util.*;
import java.time.LocalDate;

public class Donacion {
    // Atributos de la clase
    private float cantidad;
    private LocalDate fecha;

    // Constructor
    public Donacion(float cantidad, LocalDate fecha) {
        assert(fecha != null && cantidad > 0); // Nos aseguramos de que
ninguno de los parámetros sea nulo
        this.cantidad = cantidad;
        this.fecha = fecha;
        System.out.println("La donación ha tenido lugar de manera
correcta, con la cantidad " + this.cantidad + " y la fecha: " +
this.fecha);
    }

    //----------- getters ------------

    public float getCantidad() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        return cantidad;
    }

    private LocalDate getFecha() {
        return fecha;
    }

    //----------- setters -------------

    private void setCantidad(float cantidad) {
        assert(cantidad > 0);
        this.cantidad = cantidad;
    }

    private void setFecha(LocalDate fecha) {
        assert(fecha != null);
        this.fecha = fecha;
    }
```

```
}
```

Esta clase no ha sufrido ninguna modificación en el código.

2.4. VOLUNTARIO

```
import java.util.*;

public class Voluntario {
    // Variable que representa la relación
    private Socio socio;

    // Constructor
    public Voluntario(Socio socio) {
        this.socio = socio;
    }

    public void tramitarAdopcion(Adoptante ad, Animal a) {
        assert(ad != null && a != null); // Nos aseguramos de que
ninguno de los parámetros sea nulo
        if (a.getEstado() == EstadoAnimal.DISPONIBLE) { // Comprobamos
que el animal esté disponible para ser adoptado
            Date fecha = new Date();
            Adopcion adop = new Adopcion(fecha, a , ad, this);
            ad.getAdopciones().add(adop); // Añadimos la adopción a la
lista de adopciones del socio adoptante
            a.setAdopcion(adop);
            a.setEstado(EstadoAnimal.ADOPTADO);
            socio.getRefugio().getAnimalesRefugiados().remove(a);
            System.out.println("Adopción registrada: Se ha adoptado al
animal. Estado actual del animal: " + a.getEstado());
        } else {
            System.out.println("El animal no está disponible para
adopción. Estado actual del animal: " + a.getEstado());
        }
    }

    public void registrar(Animal a) {
        socio.getRefugio().registrar(a);
    }
}
```

La clase Voluntario ahora incluye una variable socio, lo que establece una relación de composición con un socio específico. En el constructor, se pasa un objeto Socio como parámetro y se asigna a esta variable, eliminando la necesidad de herencia. En el método tramitarAdopcion(), en lugar de acceder directamente a las listas de animales del refugio usando this, ahora se utiliza socio.getRefugio().getAnimalesRefugiados() para manejar los animales refugiados, manteniendo la coherencia con el nuevo diseño basado en composición y asegurando que todas las interacciones pasen a través del socio asociado. Esto hace que la relación entre Voluntario, Socio y el refugio sea más clara.

2.5. ADOPCIÓN

```
import java.util.*;

public class Adopcion {
    // Atributo de la clase
    private Date fecha;
    // Variables que representan las relaciones
    private Animal animal;
    private Adoptante adoptante;
    private Voluntario voluntario;

    // Constructor
    public Adopcion(Date fechaAdop, Animal a, Adoptante ad, Voluntario
voluntario){
        assert(fechaAdop!=null && a!=null && ad!=null &&
voluntario!=null); // Nos aseguramos de que ninguno de los parámetros
sea nulo
        this.fecha=fechaAdop;
        this.animal=a;
        this.adoptante=ad;
        this.voluntario=voluntario;
        System.err.println("Adopcion creada.");
    }

//-------------getters---------------
    private Date getFecha(){
        return fecha;
    }
```

```
    private Animal getAnimal(){
        return animal;
    }

    private Adoptante getAdoptante(){
        return adoptante;
    }

    private Voluntario getVoluntario(){
        return voluntario;
    }
///---------------setters--------------

    private void setFecha(Date fechaAdop){
        assert(fechaAdop!=null);
        this.fecha=fechaAdop;
    }

    private void setAnimal(Animal a){
        assert(a!=null);
        this.animal=a;
    }

    private void setAdoptante(Adoptante adop){
        assert(adop!=null);
        this.adoptante=adop;
    }

    private void setVoluntario(Voluntario v){
        assert(v!=null);
        this.voluntario=v;
    }
}
```

Esta clase no ha sufrido ninguna modificación en el código.

2.6. ADOPTANTE

```
import java.util.List;
import java.time.LocalDate;
import java.util.*;
```

```java
public class Adoptante {
    // Variables que representan las relaciones
    private List<Adopcion>adopciones;
    private Socio socio;

    // Constructor
    public Adoptante (Socio socio){
        adopciones = new LinkedList<Adopcion>();
        this.socio = socio;
    }

    //----------getters/setters--------------

    public List<Adopcion> getAdopciones(){
        return adopciones;
    }

    private void setAdopciones(List<Adopcion> adop){
        assert(adop != null);
        this.adopciones = adop;
    }

    //-----------Métodos-------------

    public void adoptar(Animal a, Voluntario voluntario){
        assert(a != null && voluntario != null);  // Nos aseguramos de
que ninguno de los parámetros sea nulo
         if(a.getEstado() == EstadoAnimal.DISPONIBLE) { // Comprobamos
que el animal esté disponible para ser adoptado
            voluntario.tramitarAdopcion(this, a); // Llamamos a la
función para que el voluntario sea el encargado de tramitar la adopción
         } else {
            System.err.println("Animal no disponible para la
adopción");
         }
    }
}
```

La nueva versión de la clase Adoptante ahora incluye la variable socio, estableciendo una relación de composición entre un adoptante y su socio asociado. En el constructor, se recibe un objeto Socio y

se asigna a esta variable, eliminando la dependencia de la herencia. Esto permite que Adoptante forme parte del socio como un rol específico. Toda la funcionalidad original se mantiene, como el método adoptar(), que permite iniciar una adopción comprobando que el animal esté disponible y delegando el trámite al voluntario.

2.7. ANIMAL

```
import java.util.*;
import java.time.LocalDate;

public class Animal {
    // Atributos de la clase
    private LocalDate nacimiento;
    private EstadoAnimal estado;

    // Variables que representan las relaciones
    private Adopcion adopcion;
    private Refugio refugioRegistrado;

    // Constructor
    public Animal (LocalDate nacimiento, Refugio refugio){
        assert(nacimiento != null && refugio != null); // Nos
aseguramos de que ninguno de los parámetros sea nulo
        this.nacimiento = nacimiento;
        this.estado = null;
        this.adopcion = null;
        this.refugioRegistrado = refugio;
        System.out.println("El animal se ha creado correctamente con la
fecha de nacimiento: " + this.nacimiento + " y se ha registrado en el
refugio.");
        this.refugioRegistrado.registrar(this); // Registramos al
animal en el refugio
    }

    //----------- getters -----------

    private LocalDate getNacimiento(){
        return nacimiento;
    }

    public EstadoAnimal getEstado(){ // Visibilidad pública para que se
le pueda llamar desde otras clases
```

```
        return estado;
    }

    private Adopcion getAdopcion(){
        return this.adopcion;
    }

    private Refugio getRefugioRegistrado(){
        return this.refugioRegistrado;
    }

    //----------- setters -----------

    private void setNacimiento(LocalDate nacimiento){
        assert(nacimiento != null);
        this.nacimiento = nacimiento;
    }

    public void setEstado(EstadoAnimal estado){
        assert(estado != null);
        this.estado = estado;
    }

    public void setAdopcion(Adopcion adopcion){
        assert(adopcion != null);
        this.adopcion = adopcion;
    }

    private void setRefugioRegistrado(Refugio refugio){
        assert(refugio != null);
        this.refugioRegistrado = refugio;
    }
}
```

Esta clase no ha sufrido ninguna modificación en el código.

2.8. REFUGIO

```
import java.util.ArrayList;
import java.util.List;

public class Refugio{
```

```java
    // Atributo de la clase
    private float liquidez;

    // Variables que representan las relaciones
    private List<Animal> animalesRegistrados;
    private List<Animal> animalesRefugiados;
    private List<Socio> sociosRefugio;

    // Constructor
    public Refugio() {
        this.liquidez = 0;
        this.animalesRegistrados = new ArrayList<>();
        this.animalesRefugiados = new ArrayList<>();
        this.sociosRefugio = new ArrayList<>();
        System.out.println("Se ha creado el refugio correctamente.");
    }

//----------getters/setters--------------

    public float getLiquidez() { // Visibilidad pública para que se le
pueda llamar desde otras clases
        return liquidez;
    }

    public void setLiquidez(float liquidez) { // Visibilidad pública
para que se le pueda llamar desde otras clases
        assert(liquidez >= 0);
        this.liquidez = liquidez;
    }

    public List<Animal> getAnimalesRegistrados() { // Visibilidad
pública para que se le pueda llamar desde otras clases
        return animalesRegistrados;
    }

    public List<Animal> getAnimalesRefugiados() { // Visibilidad
pública para que se le pueda llamar desde otras clases
        return animalesRefugiados;
    }

    private List<Socio> getSociosRefugio() {
        return sociosRefugio;
    }
```

```
//-----------Métodos-------------

    public void registrar(Animal a) {
        assert(a != null);
        animalesRegistrados.add(a); // Añadimos el animal a la lista de
registrados
        animalesRefugiados.add(a); // Añadimos el animal a la lista de
refugiados
        a.setEstado(EstadoAnimal.DISPONIBLE); // Cambiamos el estado
del animal a disponible
        System.out.println("Animal registrado correctamente en el
refugio.");
    }
}
```

Esta clase no ha sufrido ninguna modificación en el código.

2.9. ESTADO ANIMAL

```
public enum EstadoAnimal {
    DISPONIBLE,
    ADOPTADO,
    ENTRATAMIENTO
}
```

2.10. MAIN

```
import java.time.LocalDate;

public class Main {
    public static void main(String[] args) throws Exception {
        // Creo varias fechas y un refugio
        LocalDate fecha = LocalDate.of(2023,11,28);
        LocalDate fecha2 = LocalDate.of(2024,5,14);
        LocalDate fecha3 = LocalDate.of(2022,12,25);
        Refugio refugio = new Refugio();
        System.out.println();
        //Creamos un animal y un adoptante
```

```
        Animal animal = new Animal(fecha2, refugio);
        Animal animal2 = new Animal(fecha3, refugio);
        System.out.println("Numero total de animales registrados en el
refugio: " + refugio.getAnimalesRegistrados().size()+"\n");

        Socio adoptante = new Socio(fecha, refugio, true, false,
false);
        Socio adoptante2 = new Socio(fecha, refugio, true, false,
false);
        Socio voluntario = new Socio(fecha, refugio, false, false,
true);
        System.out.println("Numero total de animales refugiados: " +
refugio.getAnimalesRefugiados().size());
        adoptante.getAdoptante().adoptar(animal,
voluntario.getVoluntario());
        adoptante.getAdoptante().adoptar(animal2,
voluntario.getVoluntario());
        System.out.println("Numero de adopciones del adoptante1: " +
adoptante.getAdoptante().getAdopciones().size());
        System.out.println("Numero total de animales refugiados: " +
refugio.getAnimalesRefugiados().size()+"\n");

        adoptante2.getAdoptante().adoptar(animal,
voluntario.getVoluntario());
        System.out.println("Numero de adopciones del adoptante2: " +
adoptante2.getAdoptante().getAdopciones().size()+"\n"); // Esta va a
salir null

        Socio donante = new Socio(fecha, refugio, false, true, false);
        donante.getDonante().donar(950);
        System.out.println();
        donante.getDonante().donar(50);
        System.out.println("Numero total de donaciones del donante1: "
+ donante.getDonante().getDonaciones().size());

System.out.println("\n-----------------------------------------------");
        System.out.println("Ahora vamos a probar el cambio en la
asignación de roles del socio Voluntario \n");
        Animal animal3 = new Animal(fecha3, refugio);
        Animal animal4 = new Animal(fecha2, refugio);
        System.out.println();
        donante.setRolAdoptante(true);
        System.out.println("Donante ha adoptado el rol de Adoptante");
```

```
        System.out.println(donante.getRoles()+"\n");

        donante.getAdoptante().adoptar(animal3,
voluntario.getVoluntario());
        donante.setRolAdoptante(false);
        System.out.println("Donante se ha desprendido del rol de
Adoptante");
        try {
            donante.getAdoptante().adoptar(animal4,
voluntario.getVoluntario());
        } catch (Exception e) {
            System.out.println("ERROR: Este socio no puede adoptar
porque no tiene permisos de Adoptante");
        }

        System.out.println(donante.getRoles()+"\n");
    }
}
```

2.11. SALIDA POR PANTALLA

```
Se ha creado el refugio correctamente.

El animal se ha creado correctamente con la fecha de nacimiento: 2024-05-14 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.
El animal se ha creado correctamente con la fecha de nacimiento: 2022-12-25 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.
Numero total de animales registrados en el refugio: 2

Numero total de animales refugiados: 2
Adopcion creada.
Adopción registrada: Se ha adoptado al animal. Estado actual del animal: ADOPTADO
Adopcion creada.
Adopción registrada: Se ha adoptado al animal. Estado actual del animal: ADOPTADO
Numero de adopciones del adoptante1: 2
Numero total de animales refugiados: 0

Animal no disponible para la adopción
Numero de adopciones del adoptante2: 0

Donante creado.
La donación ha tenido lugar de manera correcta, con la cantidad 950.0 y la fecha: 2024-12-02
Donación realizada. Donacion: 950.0
Nueva liquidez del refugio: 950.0

La donación ha tenido lugar de manera correcta, con la cantidad 50.0 y la fecha: 2024-12-02
Donación realizada. Donacion: 50.0
Nueva liquidez del refugio: 1000.0
Numero total de donaciones del donante1: 2
-----------------------------------------
Ahora vamos a probar el cambio en la asignación de roles del socio Voluntario

El animal se ha creado correctamente con la fecha de nacimiento: 2022-12-25 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.
El animal se ha creado correctamente con la fecha de nacimiento: 2024-05-14 y se ha registrado en el refugio.
Animal registrado correctamente en el refugio.

Donante ha adoptado el rol de Adoptante
Los roles que tiene el socio son: Donante, Adoptante

Adopcion creada.
Adopción registrada: Se ha adoptado al animal. Estado actual del animal: ADOPTADO
Donante se ha desprendido del rol de Adoptante
ERROR: Este socio no puede adoptar porque no tiene permisos de Adoptante
Los roles que tiene el socio son: Donante
```