FUNCIONES (practica)

Sobreescritura de **equals** y **hashCode** para identificar por tal cosa

ejemplo:

```java
// Redefinición de equals y hashCode por patente
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Vehiculo vehiculo = (Vehiculo) obj;
    return patente.equals(vehiculo.patente);
}

@Override
public int hashCode() {
    return patente.hashCode();
}
```

EXCEPCIONES

```java
// ==================== EXCEPCIÓN PERSONALIZADA ==========
class PatenteInvalidaException extends Exception {
    public PatenteInvalidaException(String mensaje) {
        super(mensaje);
    }
}
```

EXCEPCION POR DEFECTO:

```java
/**
 * Constructor por defecto con mensaje predeterminado
 */
public PesoInsuficienteException() {
    super(message: "Error: Un animal avícola debe tener al menos 1kg de peso");
}
```

EXCEPCION usada en CONSTRUCTOR.

```
public Avicolas(String especie, int edad, String nombre, double peso, String tipoPlumaje)
    throws PesoInsuficienteException {
  super(especie, edad, nombre, peso);

  // Validar que el animal avícola tenga al menos 1kg
  if (peso < 1.0) {
    throw new PesoInsuficienteException(
        "Error: Un animal avícola debe tener al menos 1kg de peso. Peso recibido: " + peso + "kg");
  }

  this.tipoPlumaje = tipoPlumaje;
}
```

METODOS RECURSIVOS

(1)

```
// Función recursiva para buscar vehículo por patente
private static Vehiculo buscarVehiculoRecursivo(ArrayList<Vehiculo> lista, String patente, int indice) {
    // Caso base: fin de lista
    if (indice >= lista.size()) {
        return null;
    }

    // Caso base: encontrado
    if (lista.get(indice).getPatente().equalsIgnoreCase(patente)) {
        return lista.get(indice);
    }

    // Caso recursivo
    return buscarVehiculoRecursivo(lista, patente, indice + 1);
}
```

(1.1) Este metodo se usa en **crearVehiculo()**

```
// Método 1: Crear vehículo
private static void crearVehiculo() {
    System.out.println(x: "\n=== CREAR NUEVO VEHÍCULO ===");

    try {
        String patente = leerCadena(mensaje: "Ingrese patente (ej: ABC123): ").toUpperCase();

        // Validar formato de patente
        if (!validarPatente(patente)) {
            throw new PatenteInvalidaException(mensaje: "La patente debe tener formato ABC123");
        }

        // Verificar si ya existe
        if (buscarVehiculoRecursivo(listaVehiculos, patente, indice: 0) != null) {
            System.out.println(x: "ERROR: Ya existe un vehículo con esa patente.");
            return;
        }
```

(2)

```java
// Función recursiva para contar vehículos por tipo
private static int contarPorTipoRecursivo(ArrayList<Vehiculo> lista, String tipo, int indice) {
    // Caso base
    if (indice >= lista.size()) {
        return 0;
    }

    // Contar actual + resto
    int actual = lista.get(indice).getTipo().equals(tipo) ? 1 : 0;
    return actual + contarPorTipoRecursivo(lista, tipo, indice + 1);
}
```

(2) Se usa en **contarVehiculosPorTipo()**

```java
// Método adicional: Contar vehículos por tipo usando recursión
private static void contarVehiculosPorTipo() {
    System.out.println(x: "\n=== CONTADOR DE VEHÍCULOS POR TIPO ===");
    System.out.println("1. Autos: " + contarPorTipoRecursivo(listaVehiculos, tipo: "Auto", indice: 0));
    System.out.println("2. Camiones: " + contarPorTipoRecursivo(listaVehiculos, tipo: "Camion", indice: 0));
    System.out.println("3. Motos: " + contarPorTipoRecursivo(listaVehiculos, tipo: "Moto", indice: 0));
}
```

OPERACIONES CRUD

CREAR

CASO VEHICULOS

Ejemplo 1

```java
private static void crearVehiculo() {
    System.out.println(x: "\n=== CREAR NUEVO VEHÍCULO ===");

    try {
        String patente = leerCadena(mensaje: "Ingrese patente (ej: ABC123): ").toUpperCase();

        // Validar formato de patente
        if (!validarPatente(patente)) {
            throw new PatenteInvalidaException(mensaje: "La patente debe tener formato ABC123");
        }

        // Verificar si ya existe
        if (buscarVehiculoRecursivo(listaVehiculos, patente, indice: 0) != null) {
            System.out.println(x: "ERROR: Ya existe un vehículo con esa patente.");
            return;
        }

        String marca = leerCadena(mensaje: "Ingrese marca: ");
        String modelo = leerCadena(mensaje: "Ingrese modelo: ");
        int anio = leerEntero(mensaje: "Ingrese año: ");
        int kilometraje = leerEntero(mensaje: "Ingrese kilometraje: ");

        System.out.println(x: "\nSeleccione tipo de vehículo:");
        System.out.println(x: "1. Auto");
        System.out.println(x: "2. Camión");
        System.out.println(x: "3. Moto");
        int tipo = leerEntero(mensaje: "Opción: ");

        Vehiculo nuevoVehiculo = null;

        switch (tipo) {
            case 1:
                int puertas = leerEntero(mensaje: "Ingrese número de puertas: ");
                nuevoVehiculo = new Auto(marca, modelo, anio, patente, kilometraje, puertas);
                break;
            case 2:
                double carga = leerDecimal(mensaje: "Ingrese capacidad de carga (toneladas): ");
                nuevoVehiculo = new Camion(marca, modelo, anio, patente, kilometraje, carga);
                break;
            case 3:
                int cilindrada = leerEntero(mensaje: "Ingrese cilindrada (cc): ");
                nuevoVehiculo = new Moto(marca, modelo, anio, patente, kilometraje, cilindrada);
                break;
            default:
                System.out.println(x: "Tipo inválido.");
                return;
        }

        listaVehiculos.add(nuevoVehiculo);
        System.out.println(x: "\n✓ Vehículo creado exitosamente!");

    } catch (PatenteInvalidaException e) {
        System.out.println("ERROR: " + e.getMessage());
    } catch (Exception e) {
        System.out.println(x: "ERROR: Ocurrió un problema al crear el vehículo.");
    } finally {
        System.out.println(x: "Operación de creación finalizada.");
    }
}
```

Ejemplo 2

```java
public static void registrarVehiculo() {
    try {
        System.out.print(s: "Ingrese Patente (Ej: AA123BB): ");
        String patente = sc.nextLine();

        // Validación de lógica de negocio
        if (patente.length() < 6 || !patente.matches(regex: "[A-Za-z0-9]+")) {
            throw new PatenteInvalidaException(mensaje: "Formato de patente incorrecto."); //
        }

        // Lambda AnyMatch para verificar duplicados
        boolean existe = flota.stream().anyMatch(v -> v.getPatente().equalsIgnoreCase(patente));
        if (existe) {
            System.out.println(x: "Error: Ya existe un vehículo con esa patente.");
            return;
        }

        System.out.print(s: "Ingrese Marca: ");
        String marca = sc.nextLine();
        System.out.print(s: "Ingrese Modelo: ");
        String modelo = sc.nextLine();
        System.out.print(s: "Ingrese Año: ");
        int anio = Integer.parseInt(sc.nextLine());
        System.out.print(s: "Ingrese Kilometraje: ");
        double km = Double.parseDouble(sc.nextLine());

        System.out.print(s: "Tipo (1: Auto, 2: Camion): ");
        int tipo = Integer.parseInt(sc.nextLine());

            if (tipo == 1) {
                System.out.print(s: "Cantidad de puertas: ");
                int puertas = Integer.parseInt(sc.nextLine());
                flota.add(new Auto(marca, modelo, anio, patente, km, puertas));
            } else if (tipo == 2) {
                System.out.print(s: "Capacidad de carga (ton): ");
                double carga = Double.parseDouble(sc.nextLine());
                flota.add(new Camion(marca, modelo, anio, patente, km, carga));
            } else {
                System.out.println(x: "Tipo de vehículo no válido.");
            }

            System.out.println(x: "Vehículo registrado exitosamente.");

        } catch (PatenteInvalidaException e) {
            System.out.println("Error de validación: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.out.println(x: "Error: Dato numérico inválido.");
        }
    }
```

En este caso se uso **Lambda AnyMatch** para verificar duplicados (pero puede ser una funcion aparte).

CASO VETERINARIA

Funciones en la clase **Inventario** que sirven para añadir a la lista si no existe (retorna true) y si existe retorna *false*.

```
public boolean agregarAvicola(Avicolas avicola) {

  if (!avicolas.stream().anyMatch(a -> a.getNombre().equals(avicola.getNombre()))) {
    avicolas.add(avicola);
    return true;
  }
  return false; // Ya existe un animal avícola con ese nombre
}
public boolean agregarCacera(Caceras cacera) {
  if (!caceras.stream().anyMatch(c -> c.getNombre().equals(cacera.getNombre()))) {
    caceras.add(cacera);
    return true;
  }
  return false; // Ya existe un animal cazador con ese nombre
}
```

Estas 2 funciones son las que van el **Main** que sirven para cargar los datos para crear el objeto, pasar el objeto a las funciones anteriores para cargarlo y avisar si se cargo o no.

```java
private static void agregarAvicola(Inventario inventario, Scanner scanner) {
    System.out.println(x: "\n📝 AGREGAR NUEVO ANIMAL AVÍCOLA");
    System.out.print(s: "Ingrese nombre: ");
    String nombre = scanner.nextLine();
    System.out.print(s: "Ingrese especie: ");
    String especie = scanner.nextLine();
    System.out.print(s: "Ingrese edad: ");
    int edad = scanner.nextInt();
    System.out.print(s: "Ingrese peso (kg): ");
    double peso = scanner.nextDouble();
    System.out.print(s: "Ingrese tipo de plumaje (EXOTICO/COLORIDO/COMUN): ");
    scanner.nextLine(); // Limpiar buffer
    String tipoPlumaje = scanner.nextLine();


    try {
        Avicolas avicola = new Avicolas(especie, edad, nombre, peso, tipoPlumaje);

        if (inventario.agregarAvicola(avicola)) {
            System.out.println(x: "✅ Animal avícola agregado exitosamente");
        } else {
            System.out.println(x: "❌ Error: Ya existe un animal con ese nombre");
        }
    } catch (PesoInsuficienteException e) {
        System.out.println("❌ Error: " + e.getMessage());
    }
}
```

```java
private static void agregarCacera(Inventario inventario, Scanner scanner) {
    System.out.println(x: "\n📝 AGREGAR NUEVO ANIMAL CAZADOR");
    System.out.print(s: "Ingrese nombre: ");
    String nombre = scanner.nextLine();
    System.out.print(s: "Ingrese especie: ");
    String especie = scanner.nextLine();
    System.out.print(s: "Ingrese edad: ");
    int edad = scanner.nextInt();
    System.out.print(s: "Ingrese peso (kg): ");
    double peso = scanner.nextDouble();
    scanner.nextLine(); // Limpiar buffer

    Caceras cacera = new Caceras(especie, edad, nombre, peso);

    if (inventario.agregarCacera(cacera)) {
        System.out.println(x: "✅ Animal cazador agregado exitosamente");
    } else {
        System.out.println(x: "❌ Error: Ya existe un animal con ese nombre");
    }
}
```

READ O LISTAR

EJEMPLO VEHICULOS

CLAUDE

```java
// Método 2: Listar vehículos usando lambda forEach
private static void listarVehiculos() {
    System.out.println(x: "\n=== LISTA DE VEHÍCULOS ===");

    if (listaVehiculos.isEmpty()) {
        System.out.println(x: "No hay vehículos registrados.");
        return;
    }

    // Uso de interfaz lambda: forEach
    listaVehiculos.forEach(v -> {
        System.out.println(x: "\n--------------------");
        v.mostrarInfo(detallado: true);
        System.out.printf(format: "Costo mantenimiento: $%.2f%n", v.calcularCostoMantenimiento());
    });

    System.out.println("\nTotal de vehículos: " + listaVehiculos.size());
}
```

GEMINI

```java
// ----------------------------------------------------------
// MÉTODO 2: LEER (Read) con Lambda
// ----------------------------------------------------------
public static void listarVehiculos() {
    if (flota.isEmpty()) {
        System.out.println(x: "La flota está vacía.");
    } else {
        System.out.println(x: "--- LISTADO DE FLOTA ---");
        // Uso de forEach (Lambda) para recorrer la colección
        flota.forEach(v -> v.mostrarDetalles());
    }
}
```

(Mas simple Gemini)

En los 2 casos usa **lambda forEach** para listar cada uno de los objetos.

EJEMPLO VETERINARIA

```java
// Listar todos los animales
public void listarTodosLosAnimales() {
  System.out.println(x: "=== INVENTARIO DE ANIMALES ===\n");

  System.out.println("\nAVÍCOLAS (" + avicolas.size() + "):");
  avicolas.forEach(avicola -> System.out.println(avicola.toString()));

  System.out.println("\nCAZADORES (" + caceras.size() + "):");
  caceras.forEach(cacera -> System.out.println(cacera.toString()));

  System.out.println("\nTotal de animales: " + getCantidadDeAnimales());
}
```

BUSCAR

EJEMPLO VEHICULOS

Claude

```java
private static void buscarVehiculoPorPatente() {
    System.out.println(x: "\n=== BUSCAR VEHÍCULO ===");
    String patente = leerCadena(mensaje: "Ingrese patente a buscar: ").toUpperCase();

    Vehiculo encontrado = buscarVehiculoRecursivo(listaVehiculos, patente, indice: 0);

    if (encontrado != null) {
        System.out.println(x: "\n✓ Vehículo encontrado:");
        encontrado.mostrarInfo(detallado: true);
        System.out.printf(format: "Costo mantenimiento: $%.2f%n", encontrado.calcularCostoMantenimiento());
    } else {
        System.out.println(x: "✗ No se encontró ningún vehículo con esa patente.");
    }
}
// Función recursiva para buscar vehículo por patente
private static Vehiculo buscarVehiculoRecursivo(ArrayList<Vehiculo> lista, String patente, int indice) {
    // Caso base: fin de lista
    if (indice >= lista.size()) {
        return null;
    }

    // Caso base: encontrado
    if (lista.get(indice).getPatente().equalsIgnoreCase(patente)) {
        return lista.get(indice);
    }

    // Caso recursivo
    return buscarVehiculoRecursivo(lista, patente, indice + 1);
}
```

En claude se usó un metodo recursivo para buscar por patente.

Gemini

```java
// ------------------------------------------------------------
// MÉTODO 3: BUSCAR (Recursivo)
// ------------------------------------------------------------
public static void buscarVehiculoInterfaz() {
    System.out.print(s: "Ingrese la patente a buscar: ");
    String patenteBuscada = sc.nextLine();

    // Llamada al método recursivo
    int indice = buscarRecursivo(patenteBuscada, indice: 0);

    if (indice != -1) {
        System.out.println("Vehículo encontrado en el índice: " + indice);
        flota.get(indice).mostrarDetalles();
    } else {
        System.out.println(x: "Vehículo no encontrado.");
    }
}

// Función recursiva auxiliar
private static int buscarRecursivo(String patente, int indice) {
    // Caso base: llegamos al final de la lista
    if (indice >= flota.size()) {
        return -1;
    }
    // Caso base: encontramos el elemento
    if (flota.get(indice).getPatente().equalsIgnoreCase(patente)) {
        return indice;
    }
    // Llamada recursiva: buscar en el siguiente índice
    return buscarRecursivo(patente, indice + 1);
}
```

En *Gemini* hizo lo mismo.

Por lo tanto para la funcion **BUSCAR** por tal cosa, conviene tener una **funcion recursiva auxiliar.**

EJEMPLO VETERINARIA

Esta es una funcion de la clase **inventario** que segun el nombre como parametro, busca en las dos listas (avicolas y caceras) y devuelve el objeto(Animalito) si lo encuentra o retorna NULL.

```java
public Animalito buscarAnimal(String nombre) {

  for (Avicolas avicola : avicolas) {
    if (avicola.getNombre().equals(nombre)) {
      return avicola;
    }
  }

  for (Caceras cacera : caceras) {
    if (cacera.getNombre().equals(nombre)) {
      return cacera;
    }
  }

  return null;

}
```

Esta funcion va dentro del **Main** que sirve para pedir el nombre, crear un objeto de tipo Animalito y pasarlo a la funcion anterior para que esta devuelva el objeto o null y asi podes avisar si se encontro o no.

```java
private static void buscarAnimal(Inventario inventario, Scanner scanner) {
  System.out.print(s: "🔍 Ingrese nombre del animal a buscar: ");
  String nombre = scanner.nextLine();

  Animalito animal = inventario.buscarAnimal(nombre.toUpperCase());
  if (animal != null) {
    System.out.println(x: "✅ Animal encontrado:");
    String emoji = animal.verTipoDeAnimal();
    System.out.println(emoji + " " + animal);
  } else {
    System.out.println(x: "❌ No se encontró un animal con ese nombre");
  }
}
```

MODIFICAR

EJEMPLO VEHICULOS

Claude

```java
// Método 4: Modificar vehículo (demuestra paso por referencia)
private static void modificarVehiculo() {
    System.out.println(x: "\n=== MODIFICAR VEHÍCULO ===");
    String patente = leerCadena(mensaje: "Ingrese patente del vehículo a modificar: ").toUpperCase();

    Vehiculo vehiculo = buscarVehiculoRecursivo(listaVehiculos, patente, indice: 0);

    if (vehiculo == null) {
        System.out.println(x: "✗ Vehículo no encontrado.");
        return;
    }

    System.out.println(x: "\nVehículo actual:");
    vehiculo.mostrarInfo(detallado: true);

    System.out.println(x: "\n¿Qué desea modificar?");
    System.out.println(x: "1. Marca");
    System.out.println(x: "2. Modelo");
    System.out.println(x: "3. Año");
    System.out.println(x: "4. Kilometraje");
    int opcion = leerEntero(mensaje: "Opción: ");

    // Paso por referencia: modificamos el objeto directamente
    modificarAtributoVehiculo(vehiculo, opcion);

    System.out.println(x: "\n✓ Vehículo modificado exitosamente!");
    vehiculo.mostrarInfo(detallado: true);
}
```

```java
// Método que modifica objeto (paso por referencia)
private static void modificarAtributoVehiculo(Vehiculo v, int opcion) {
    switch (opcion) {
        case 1:
            String marca = leerCadena(mensaje: "Nueva marca: ");
            v.setMarca(marca);
            break;
        case 2:
            String modelo = leerCadena(mensaje: "Nuevo modelo: ");
            v.setModelo(modelo);
            break;
        case 3:
            int anio = leerEntero(mensaje: "Nuevo año: ");
            v.setAnio(anio);
            break;
        case 4:
            int km = leerEntero(mensaje: "Nuevo kilometraje: ");
            v.setKilometraje(km);
            break;
        default:
            System.out.println(x: "Opción inválida.");
    }
}
```

Para modificar un atributo de un objeto vamos a usar tmb 2 funciones usando el paso por referencia.

1. En la primer funcion preguntamos la identificacion del objeto
2. **Segundo**: buscamos si existe; si existe nos devuelve ese objeto.
3. **Tercero**: preguntamos que modificar.
4. **Cuarto**: usamos la funcion para modificar tal atributo parametrizandolo con el objeto y la opcion(int).
5. **Quinto**: mostamos la info del objeto actualizado.

EJEMPLO VETERINARIA

Ambas son funciones de Inventario que reciben el nombre y el objeto Avicolas o Caceras(con sus nuevos datos) como parametros.

Dentro de la funcion se busca dentro de la lista al objeto que tenga el mismo nombre. Si lo encuentra, lo setea y retorna TRUE(para confirmar la operacion).

Si no lo encuentra por el nombre retorna FALSE.

Avicola

```java
public boolean actualizarAvicola(String nombre, Avicolas avicolaActualizada) {
  for (int i = 0; i < avicolas.size(); i++) {
    if (avicolas.get(i).getNombre().equals(nombre)) {
      avicolas.set(i, avicolaActualizada);
      return true;
    }
  }
  return false;
```

Cacera

```java
// Actualizar Cacera
public boolean actualizarCacera(String nombre, Caceras caceraActualizada) {
  for (int i = 0; i < caceras.size(); i++) {
    if (caceras.get(i).getNombre().equals(nombre)) {
      caceras.set(i, caceraActualizada);
      return true;
    }
  }
  return false;
}
```

Estas dos funciones dentro del **MAIN** van a pedir los datos.
Van a buscar el animal usando inventario.buscarAnimal(), si no existe printea que no se encontro pero si existe va a pedir los datos y va crear un objeto de tipo Avicolas/Caceras con todos los datos nuevos.
Despues va a llamar a la funcion actualizarAvicola/actualizarCacera() para actualizar realmente el objeto en la lista.

```java
private static void actualizarAvicola(Inventario inventario, Scanner scanner) {
    System.out.print(s: "🔄 Ingrese nombre del animal avícola a actualizar: ");
    String nombre = scanner.nextLine();

    if (inventario.buscarAnimal(nombre.toUpperCase()) != null) {
        System.out.println(x: "Ingrese los nuevos datos:");
        System.out.print(s: "Especie: ");
        String especie = scanner.nextLine();
        System.out.print(s: "Edad: ");
        int edad = scanner.nextInt();
        System.out.print(s: "Peso (kg): ");
        double peso = scanner.nextDouble();
        System.out.print(s: "Tipo de plumaje: ");
        scanner.nextLine(); // Limpiar buffer
        String tipoPlumaje = scanner.nextLine();

        try {
            Avicolas avicolaActualizada = new Avicolas(especie, edad, nombre, peso, tipoPlumaje);

            if (inventario.actualizarAvicola(nombre.toUpperCase(), avicolaActualizada)) {
                System.out.println(x: "✅ Animal avícola actualizado exitosamente");
            } else {
                System.out.println(x: "❌ Error al actualizar el animal avícola");
            }
        } catch (PesoInsuficienteException e) {
            System.out.println("❌ Error: " + e.getMessage());
        }
    } else {
        System.out.println(x: "❌ No se encontró un animal avícola con ese nombre");
    }
}
```

```java
private static void actualizarCacera(Inventario inventario, Scanner scanner) {
    System.out.print(s: "🔄 Ingrese nombre del animal cazador a actualizar: ");
    String nombre = scanner.nextLine();

    if (inventario.buscarAnimal(nombre.toUpperCase()) != null) {
        System.out.println(x: "Ingrese los nuevos datos:");
        System.out.print(s: "Especie: ");
        String especie = scanner.nextLine();
        System.out.print(s: "Edad: ");
        int edad = scanner.nextInt();
        System.out.print(s: "Peso (kg): ");
        double peso = scanner.nextDouble();
        scanner.nextLine(); // Limpiar buffer

        Caceras caceraActualizada = new Caceras(especie, edad, nombre, peso);

        if (inventario.actualizarCacera(nombre.toUpperCase(), caceraActualizada)) {
            System.out.println(x: "✅ Animal cazador actualizado exitosamente");
        } else {
            System.out.println(x: "❌ Error al actualizar el animal cazador");
        }
    } else {
        System.out.println(x: "❌ No se encontró un animal cazador con ese nombre");
    }
}
```

DELETE/ELIMINAR

EJEMPLO VEHICULOS

Claude

```java
// Método 5: Eliminar vehículo usando lambda removeIf
private static void eliminarVehiculo() {
    System.out.println(x: "\n=== ELIMINAR VEHÍCULO ===");
    String patente = leerCadena(mensaje: "Ingrese patente del vehículo a eliminar: ").toUpperCase();

    // Uso de interfaz lambda: removeIf
    boolean eliminado = listaVehiculos.removeIf(v -> v.getPatente().equalsIgnoreCase(patente));

    if (eliminado) {
        System.out.println(x: "✓ Vehículo eliminado exitosamente!");
    } else {
        System.out.println(x: "✗ No se encontró ningún vehículo con esa patente.");
    }
}
```

En este caso elimina por patente.

Gemini

```java
// ----------------------------------------------------------
// MÉTODO 5: ELIMINAR (Delete) con Lambda
// ----------------------------------------------------------
public static void eliminarPorCondicion() {
    System.out.print(s: "Eliminar vehículos con más de X km. Ingrese X: ");
    try {
        double limite = Double.parseDouble(sc.nextLine());

        // Uso de removeIf (Lambda) para eliminar condicionalmente
        boolean eliminado = flota.removeIf(v -> v.getKilometraje() > limite);

        if (eliminado) {
            System.out.println(x: "Se eliminaron los vehículos que cumplían la condición.");
        } else {
            System.out.println(x: "Ningún vehículo cumplía la condición.");
        }
    } catch (Exception e) {
        System.out.println(x: "Error al ingresar el límite.");
    }
}
```

Acá elimina por kilometraje.

En DELETE/ELIMINAR se usa **Lambda removeIf**

EJEMPLO VETERINARIA

Funcion que esta dentro de la clase **inventario** que recibe el nombre como parametro.
Crea dos variables booleanas (una por cada especie) y los remueve de la lista segun el nombre usando **lista.removeIf(lambda)** que va a retonar false o true.

```java
public boolean eliminarAnimal(String nombre) {
  // Intentar eliminar de avícolas
  boolean eliminadoDeAvicolas = avicolas.removeIf(avicola -> avicola.getNombre().equals(nombre));
  boolean eliminadoDeCaceras = caceras.removeIf(cacera -> cacera.getNombre().equals(nombre));
  if (eliminadoDeAvicolas || eliminadoDeCaceras) {
    return true;
  }
  return eliminadoDeCaceras;
}
```

Esta funcion esta en el **Main** y toma el nombre del animal a eliminar.
Llama a la funcion anterior que va a retornar true o false.

```java
private static void eliminarAnimal(Inventario inventario, Scanner scanner) {
    System.out.print(s: "🗑️ Ingrese nombre del animal a eliminar: ");
    String nombre = scanner.nextLine();

    if (inventario.eliminarAnimal(nombre.toUpperCase())) {
        System.out.println(x: "✅ Animal eliminado exitosamente");
    } else {
        System.out.println(x: "❌ No se encontró un animal con ese nombre");
    }
}
```

FILTER LAMBDA

```java
// Método adicional: Calcular costos usando filter (lambda)
private static void calcularCostosMantenimiento() {
    System.out.println(x: "\n=== COSTOS DE MANTENIMIENTO ===");

    if (listaVehiculos.isEmpty()) {
        System.out.println(x: "No hay vehículos registrados.");
        return;
    }

    // Uso de streams con filter (interfaz lambda)
    double costoTotal = listaVehiculos.stream()
        .mapToDouble(Vehiculo::calcularCostoMantenimiento)
        .sum();

    System.out.printf(format: "Costo total de mantenimiento: $%.2f%n", costoTotal);

    // Mostrar vehículos con mantenimiento > 1000
    System.out.println(x: "\nVehículos con mantenimiento mayor a $1000:");
    listaVehiculos.stream()
        .filter(v -> v.calcularCostoMantenimiento() > 1000)
        .forEach(v -> {
            System.out.printf(format: "- %s (Patente: %s): $%.2f%n",
                v.getTipo(), v.getPatente(), v.calcularCostoMantenimiento());
        });
}
```