

PAC 4. UF5.

Control de excepciones. Lectura y escritura de información.

Los puntos a realizar para el control de excepciones, lectura y escritura de información son los siguientes:

- La aplicación deberá confirmar que el DNI introducido es válido.
- Almacenaremos el ArrayList \diamond de clientes en un fichero de texto y lo recuperaremos al iniciar la aplicación.

En primer lugar crearemos un método para validar el DNI denominado `validaDNI` al que le pasaremos el DNI como String.

Comprobaremos la longitud del DNI y la letra correspondiente. Además utilizaremos un array de tipo `char` para recorrer las letras del DNI. Y por último convertiremos el DNI en un número entero, calculado el resto, para saber si de las letras que tenemos, la introducida es la correcta.

```
private boolean validaDNI(String dni) {  
    // numeros y 1 letra (longitud 9)  
    if (dni.length() != 9) {  
        System.out.println("El DNI no tiene caracteres correctos");  
        return false;  
    }  
    else if (Character.isLetter(dni.charAt(8)) == false) {  
        System.out.println("El DNI no tiene letra");  
        return false;  
    }  
    }  
    char [] letrasDNI = {'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};  
    int numero = Integer.parseInt(dni.substring(0,8)); //convertimos el dni en entero  
    // calculamos el resto  
    int resto = numero%23;  
    if (letrasDNI[resto] == dni.toUpperCase().charAt(8)){  
        return true; //la letra es correcta  
    }  
    else {  
        return false; //la letra no es correcta  
    }  
}
```

Para comprobar la validez del DNI, deberemos adaptar determinadas partes del código donde se extraiga el DNI. Las partes referentes son las siguientes:

```
String dni = reg_dniClienteText.getText();

if(validaDNI(dni) == true ) {

    Mascota mascota = null;

    if(reg_gatoRB.isSelected()) {
        mascota = new Gato();
        ((Gato) mascota).setColor(reg_colorText.getText());
    }
    else if(reg_perroRB.isSelected()) {
        mascota = new Perro();
        ((Perro) mascota).setRaza(reg_razaText.getText());
    }
    else if(reg_roedorRB.isSelected()) {
        mascota = new Roedor();
        ((Roedor) mascota).setTipo(reg_conejoRB.isSelected()?"conejo":"roedor");
    }
    String nombre = reg_nombreMascotaText.getText();
    mascota.setNombre(nombre);
    //mascota.setNombre(reg_nombreMascotaText.getText());
    mascota.setGenero(reg_machoRB.isSelected()?"macho":"hembra");

    Cliente cliente = new Cliente();

    cliente.setDni(dni);
    cliente.setNombre(reg_nombreClienteText.getText());

    cliente.addMascota(mascota);

    // añadimos el cliente a la "base de datos"
    clientes.add(cliente);

    System.out.println("Datos cliente: " + cliente);
}

else {

    System.out.println("ERROR: El DNI no es valido");
}
```

```

String dni = trat_dniClienteText.getText();

if (validaDNI(dni) == true) {

    trat_mascotasComboBox.removeAllItems();

    for (Cliente cli: clientes) {
        if (cli.getDni().equalsIgnoreCase(dni)) {
            for (Mascota m : cli.getMascotas()) {
                String nombreMascota = m.getNombre();
                //trat_mascotas.add(nombreMascota);
                trat_mascotasComboBox.addItem(nombreMascota);
            }
            trat_mascotasComboBox.setEnabled(true);
            break;
        }
    }

}

else {
    System.out.println("ERROR: El DNI no es valido");
}

```

```

String dni = trat_dniClienteText.getText();

if (validaDNI(dni) == true) {

    for (Cliente cli: clientes) {
        if (cli.getDni().equalsIgnoreCase(dni)) {
            for (Mascota m : cli.getMascotas()) {
                String nombreMascota = m.getNombre();
                //trat_mascotas.add(nombreMascota);
                String fecha = trat_fechaText.getText();
                String tratamiento = trat_tratamientoText.getText();
                m.addTratamiento(fecha, tratamiento);
                System.out.println(m.getNombre() + " :: fecha: " + fecha + " - tratamiento: " + tratamiento);
            }
            trat_mascotasComboBox.setEnabled(true);
            break;
        }
    }

}

else {
    System.out.println("ERROR: El DNI no es valido");
}

```

Lo que hemos realizado en esta parte del código es controlar los posibles errores que ha podido llevar a cabo el usuario imprimiéndolos por consola. Como queda demostrado en el código y en la ejecución del programa.

Hasta aquí hemos realizado la parte de creación del algoritmo de introducción del DNI válido.

Para almacenar el ArrayList de clientes en un fichero de texto, deberemos primero, poder guardar los objetos java en el disco de almacenamiento. Esta acción la llevaremos a través de un flujo o stream (del mismo modo funciona como en otros lenguajes como C#).

Además para que un objeto pueda ser transportado a través del flujo o stream, este objeto tendrá que ser serializable. Por lo que en primer lugar serializaremos la clase.

```
package VetIlerna;

import java.io.Serializable;

//paquete

public class Cliente implements Serializable { //clase serializamos la clase UF5_PAC4

    /* para poder guardar objetos java en disco, se ha de realizar a traves de un flujo o stream
     * para que un objeto pueda ser transportado a través de un stream, este objeto debe ser serializable
     */

    private String nombre, dni; //atributos o propiedades de la clase
    private ArrayList<Mascota> mascotas;

    public Cliente() { //constructor vacio que no recibe parametros
        mascotas = new ArrayList<>();
    }

    public Cliente (String nombre,String dni) // constructor con parametros de tipo string
    {
        this.nombre=nombre; //enlace de atributos de la clase con los parámetros del constructor
        this.dni=dni;
        mascotas = new ArrayList<>();
    }
}
```

```

package VetIlerna; //paquete

import java.io.Serializable;

public class Mascota implements Serializable { //clase

    /* para poder guardar objetos java en disco, se ha de realizar a traves de un flujo o stream
    * para que un objeto pueda ser transportado a través de un stream, este objeto debe ser serializable
    * tambien es imprescindible que todas las propiedades de una clase serializable sean a su vez serializables
    */

    private String nombre, genero; //atributos o propiedades de la clase
    private static int contador = 1;
    private int codigo;
    private HashMap<String,String> tratamientos; // <fecha, tratamiento>

    public Mascota() { //constructor vacio que no recibe parametros
        codigo = contador++;
        tratamientos = new HashMap<>();
    }

    public Mascota(String nombre, String genero) // constructor con parametros de tipo string e int
    {
        codigo = contador++;
        this.nombre=nombre; //enlace de atributos de la clase con los parámetros del constructor
        this.genero=genero;
        tratamientos = new HashMap<>();
    }
}

```

Esta primera acción la hemos realizado en las clases Mascota y Cliente.

A continuación llevaremos a cabo la creación del ArrayList<> de escritura y lectura.

En primer lugar comprobaremos si existe el fichero, previamente declararemos el nombre del mismo que se almacenará en la ruta correspondiente (las imágenes se muestran a continuación):

```

public static void main(String[] args) {

    String ruta = "vetIlerna.bin";

    //ArrayList<Cliente> clientes = new ArrayList<>();

    // comprobamos primero si esta el fichero (si ya existe)

    ArrayList<Cliente> clientes = leerEnDiscoBinario(ruta);

    if (clientes == null) {
        clientes = new ArrayList<Cliente>();
    }

    VentanaDefinitiva v = new VentanaDefinitiva(clientes);

    /*
     * Antes de cerrar el programa guardamos los datos en el disco
     *
     * */

    escribirEnDiscoBinario(clientes, ruta);

}

```

Posteriormente diferenciaremos entre lectura y escritura:

```

//Escritura
private static void escribirEnDiscoBinario(ArrayList<Cliente> clientes, String ruta) {

    try {

        FileOutputStream fos = new FileOutputStream(ruta);
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        oos.writeObject(clientes);
        oos.close();

    }

    catch (IOException e) {

        System.out.println("ERROR: El fichero no existe");
        e.printStackTrace();

    }

}

```

```
//Lectura
private static ArrayList<Cliente> leerEnDiscoBinario(String ruta) {

    try(FileInputStream fis = new FileInputStream(ruta);
        ObjectInputStream ois = new ObjectInputStream(fis);
        )
    /*
     * Esto es un try con recursos de forma que cuando termina o falla cierra los recursos automaticamente
     * No necesitamos cerrarlos manualmente
     */

    {
        return (ArrayList<Cliente>)ois.readObject();
    } catch( IOException | ClassNotFoundException e ) {

        if (e instanceof FileNotFoundException) {

            //if (e.getMessage().indexOf("No existe") != -1 ) {

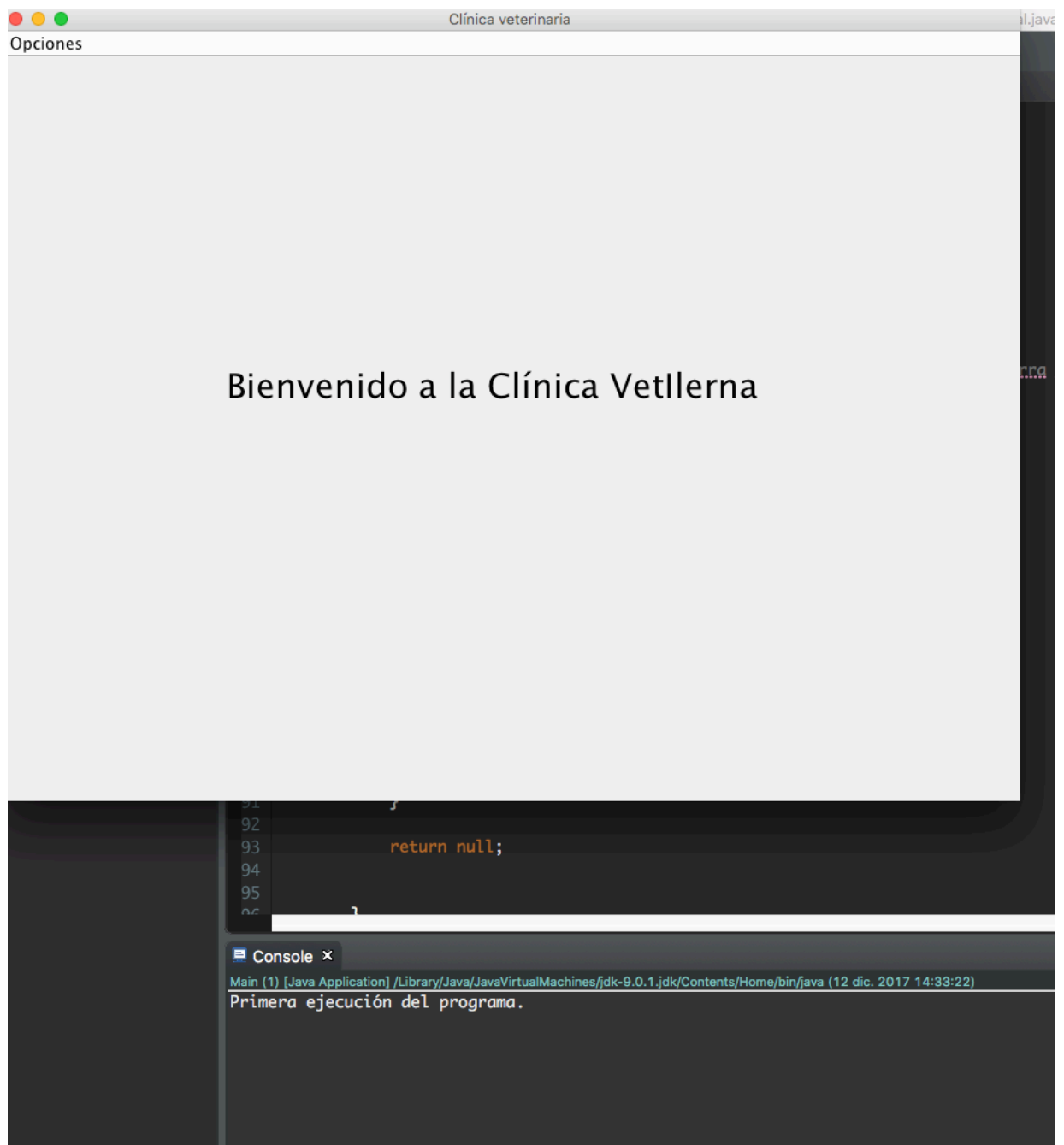
                System.out.println("Primera ejecución del programa.");


            }else {
                e.printStackTrace();
            }

        }

        return null;
    }
}
```

El programa realizará una comprobación de ejecución la primera vez, porque no encontrará el fichero, cuando lo iniciemos la segunda vez esto no nos volverá a aparecer por consola.



▶		bin	✓	ayer 21:20
▶		src	✓	ayer 21:19
		vetllerna.bin	✓	hoy 14:33

Clínica veterinaria

Opciones

Registro Cliente

Anibal Santos 70873352W

Dirección Cliente

Datos mascota

Congo

Tipo

☐ Gato ☒ Perro ☐ Roedor

color Galgo

macho hembra

Especialización de Roedor

☐ Conejo ☐ Ratón

Insertar

```
91
92
93     return null;
94
95
96
```

Console x

Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk-9.0.1.jdk/Contents/Home/bin/java (12 dic. 2017 14:36:07)

Datos cliente: Anibal Santos con DNI: 70873352W tiene las siguientes mascotas:
Congo con el código 1 es macho tratamientos: {} y es un perro de raza Galgo

Aquí hemos comprobado el registro como cliente.

Y en las siguientes capturas vemos el registro y el control de excepciones pertinente a la introducción errónea de datos.

