Manuel Sánchez Salazar

Descripción general

Este proyecto es una aplicación de escritorio desarrollada en Java utilizando Java Swing. el objetivo es gestionar un registro de elementos (en este coches) haciendo las operaciones CRUD y proporcionar una experiencia completamente internacionalizada al usuario.

Los datos se almacenan en un archivo de texto plano (`TSV`) y se cargan en memoria al iniciar la aplicación.

La interfaz permite cambiar entre idiomas lo que se refleja tanto en los textos como en las imágenes asociadas. La aplicación utiliza una estructura modular para mantener separadas las responsabilidades del manejo de datos, la lógica del negocio y la interfaz gráfica.

Decisiones clave

Estructura del Proyecto

Separación de Responsabilidades

Se han implementado diferentes clases para cada funcionalidad clave

- `CocheController`: Encargada de gestionar la lógica relacionada con los coches, incluyendo las operaciones CRUD
- `IdiomaController`: Responsable de cargar, gestionar y proveer cadenas de texto y rutas de imágenes según el idioma seleccionado.
- Ventanas individuales para cada operación, que simplifica la interacción del usuario y mantener un diseño limpio.
- Estas ventanas se comunican con el `CocheController` y el `IdiomaController`

Uso de Estructuras de Datos:

- Los coches se almacenan en un `Map<String, Coche>`, utilizando la matrícula como clave primaria. Esto asegura accesos rápidos y evita duplicados

Internacionalización

Archivo de Idiomas

- Un único archivo `ArchivoIdiomas.tsv` que contiene las cadenas de texto y las rutas de imágenes necesarias para cada idioma
- Estructura del archivo:

```
<Número de Idiomas>
<código del idioma>
<número de cadenas de texto>
<cadenas de texto>
<número de imagenes>
<rutas de imágenes>
```

Carga dinámica de idiomas

- La clase `IdiomaController` lee este archivo al iniciar la aplicación y almacena las cadenas en un `Map<String,List<String>>`, donde la clave es el código del idioma y el valor es la lista de cadenas correspondientes. Las imágenes y las banderas reciben un trato similar.
- La lista de idiomas disponibles también se carga para mostrar las opciones en el menú
- Al utilizar las cadenas, estas se obtienen usando directamente la posición en la que están en el archivo de entre las cadenas de un mismo idioma, por ejemplo:

```
fileMenu = new JMenu(cadenas.get(1)); // File o Archivo
```

Como podemos ver arriba, al crear el menú de archivo, no le damos una cadena directamente, sino que le damos la posición de la cadena en la lista, por lo que al cambiar de idioma solo hay que redibujar la aplicación.

Cambios dinámicos

- Todos los textos de todos los componentes se actualizan automáticamente al cambiar el idioma mediante un redibujado de la interfaz.

Personalización de JOptionPane:

- Para resolver el problema de que los botones en los cuadros de dialogo tengan los botones en el idioma predeterminado, se han usado las siguientes órdenes para sustituir el texto predeterminado(`Cancel` , `Ok` , `Yes` , `No`) utilizando el siguiente código:

```
UIManager.put("OptionPane.cancelButtonText", cadenas.get(14)); // "Cancelar"
UIManager.put("OptionPane.okButtonText", cadenas.get(27)); // "Vale" o "OK"
UIManager.put("OptionPane.yesButtonText", cadenas.get(24)); // "Sí" o "Yes"
UIManager.put("OptionPane.noButtonText", cadenas.get(25));  // "No" o "No"
```

- Estas cuatro ordenes son llamadas cada vez que se cambia el idioma, para así asegurarnos de que cambien estos botones.

Interfaz de Usuario

Ventanas modulares

- Cada operación CRUD tiene su propia ventana. Esto hace que cada tarea sea más clara

Tabla de Resultados

- La tabla muestra todos los coches al inicio. Al realizar una búsqueda se filtran los resultados en la tabla sin eliminar los datos originales.

Confirmación de salida

- Al intentar cerrar la aplicación, se muestra un cuadro de diálogo de confirmación personalizado para evitar cierres accidentales.