UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1
CATEDRÁTICO: ING. NEFTALI CALDERÓN
TUTOR ACADÉMICO: RODRIGO ANTONIO PORÓN DE LEÓN

MANUAL TÉCNICO

JOSUÉ DAVID VELÁSQUEZ IXCHOP
CARNÉ: 202307705
SECCIÓN: E

GUATEMALA, 1 DE ABRIL DEL 2,024

DESCRIPCIÓN GENERAL DEL PROYECTO

Se ha desarrollado un software destinado a la gestión y realización de viajes para usuarios individuales, permitiéndoles seleccionar tanto el punto de partida como el destino deseado, junto con la elección del medio de transporte más adecuado. La aplicación cuenta con una limitación de tres pilotos y una flota de nueve vehículos diferentes.

Una de las características destacadas de este programa es su capacidad para importar datos de rutas desde archivos CSV, lo que simplifica la inclusión de información detallada sobre las distintas rutas y sus respectivas distancias. Una vez cargadas estas rutas, los usuarios tienen la libertad de editarlas según sea necesario antes de proceder con los viajes planificados.

Además, el software proporciona una representación visual del estado actual del viaje, que incluye información sobre la ruta en curso, el nivel de combustible disponible y una imagen del tipo de vehículo seleccionado. En caso de que un vehículo se quede sin gasolina durante el trayecto, se ofrece un botón para recargar combustible de manera conveniente.

Al llegar al destino final, se presenta un botón que permite al usuario regresar al punto inicial si es necesario. Por último, la aplicación incluye una opción para acceder al historial completo de viajes realizados, lo que permite a los usuarios revisar y analizar sus experiencias pasadas.

Es importante destacar que el programa está diseñado para retener su estado incluso después de que se cierre, lo que significa que al reiniciarlo, continuará desde donde se dejó anteriormente, manteniendo todos los viajes en curso y cualquier edición realizada.

DIAGRAMA DE FLUJO

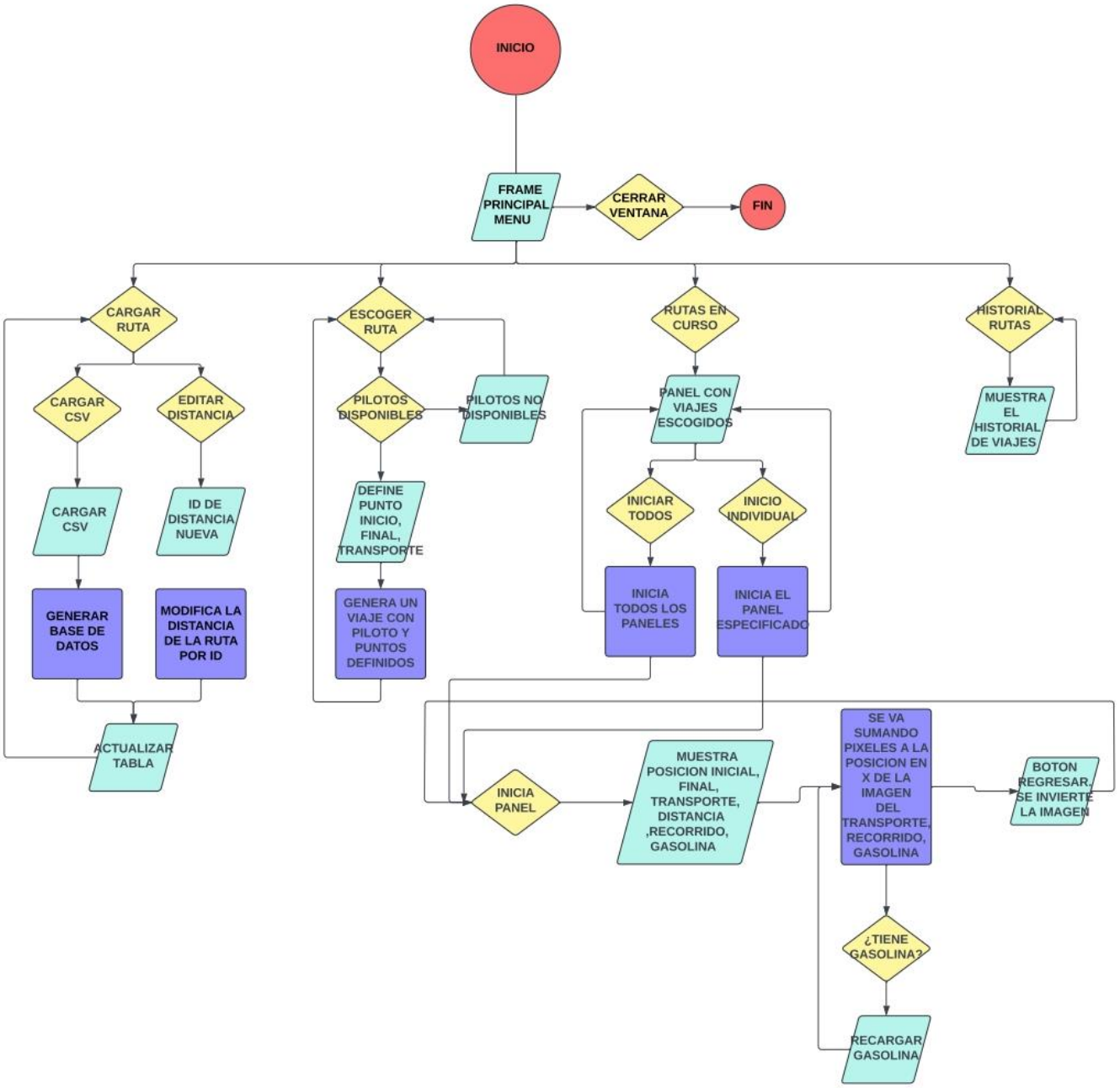
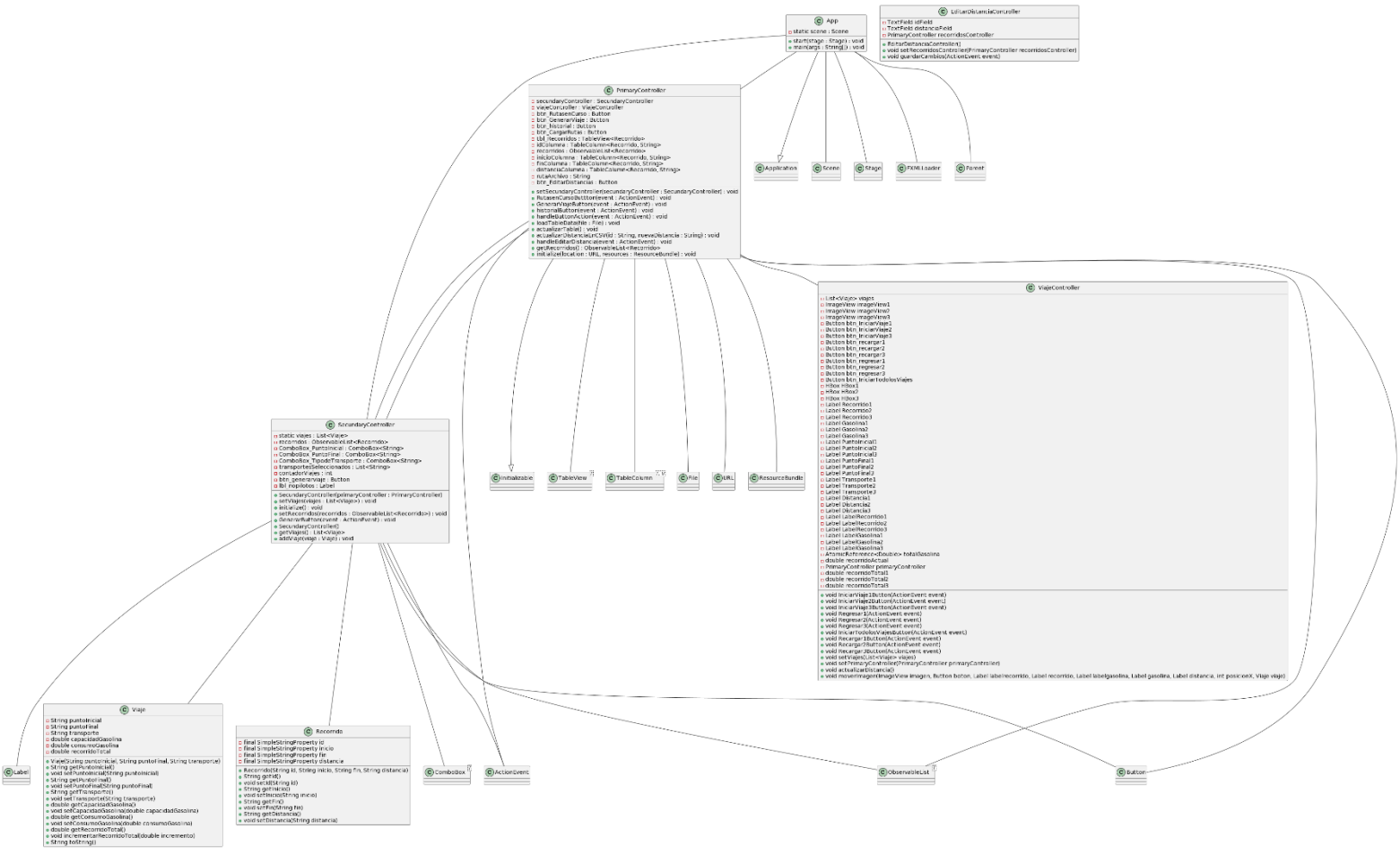


DIAGRAMA DE CLASES



LÓGICA DEL PROGRAMA

Clase App

1. Descripción

La clase **App** es la clase principal de la aplicación JavaFX. Extiende de **Application** y controla el inicio y la configuración de la interfaz gráfica de usuario (GUI).

2. Atributos

- **scene**: Un objeto de tipo **Scene** que representa la escena principal de la aplicación.

3. Métodos

Método start(Stage stage)

Este método se llama automáticamente al iniciar la aplicación y configura la interfaz gráfica. Los pasos principales son:

- **Carga de archivos FXML:** Utiliza un **FXMLLoader** para cargar los archivos FXML de **primary.fxml** y **secundary.fxml**, que contienen la estructura de la interfaz gráfica.
- **Obtención de controladores:** Utiliza el método **getController()** del **FXMLLoader** para obtener instancias de los controladores **PrimaryController** y **SecundaryController**.
- **Configuración de controladores:** Establece una referencia del controlador secundario en el controlador primario utilizando el método **setSecundaryController()**.
- **Configuración de la escena:** Crea una escena con el nodo raíz de **primary.fxml** y la configura en la ventana del **Stage**.
- **Estilo de la ventana:** Configura el estilo de la ventana del **Stage** como

UNDECORATED, lo que significa que la ventana no tendrá decoraciones de título o bordes.

- **Estilo CSS:** Agrega el archivo de estilo **style.css** a la escena para aplicar estilos adicionales.
- **Visualización de la ventana:** Muestra la ventana del **Stage** con la escena configurada.

Método main(String[] args)

Este método es la entrada principal del programa Java. Llama al método **launch()**, que a su vez inicia la aplicación JavaFX.

4. Uso

- Para iniciar la aplicación, ejecute el método **main()** de la clase **App**.
- Asegúrese de que los archivos FXML y el archivo de estilo CSS estén configurados correctamente y disponibles en las rutas especificadas.

5. Dependencias Externas

- **JavaFX Library:** Se requiere la biblioteca JavaFX para compilar y ejecutar la aplicación.

Clase PrimaryController

1. Descripción

La clase PrimaryController es un controlador de la interfaz gráfica de usuario correspondiente a la vista principal de la aplicación. Esta clase se encarga de gestionar la lógica de la GUI y manejar las interacciones del usuario con los elementos de la pantalla.

Atributos

secundaryController: Un objeto de tipo SecundaryController que representa el controlador de la vista secundaria.

viajeController: Un objeto de tipo ViajeController que representa el controlador de la vista de los viajes en curso.

rutaArchivo: Una cadena que almacena la ruta del archivo CSV cargado.

recorridos: Una lista observable de objetos Recorrido que almacena los datos de los recorridos cargados desde el archivo CSV.

Métodos

Método setSecundaryController(SecundaryController secundaryController)

Este método establece el controlador de la vista secundaria para poder acceder a él desde el controlador primario.

Método RutasenCursoButtton(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Rutas en Curso". Carga la vista de los viajes en curso y pasa los datos de los viajes desde secundaryController a viajeController.

Método GenerarViajeButton(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Generar Viaje". Carga la vista secundaria para generar un nuevo viaje.

Método historialButton(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Historial". Carga la vista del historial de viajes realizados.

Método handleButtonAction(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Cargar Rutas". Abre un FileChooser para seleccionar un archivo CSV y carga los datos de los recorridos desde ese archivo.

Método loadTableData(File file)

Este método carga los datos de los recorridos desde un archivo CSV y los muestra en una tabla en la interfaz gráfica.

Método actualizarDistanciaEnCSV(String id, String nuevaDistancia)

Este método actualiza la distancia de un recorrido en el archivo CSV y en la lista de recorridos cargados.

Método handleEditarDistancia(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Editar Distancias". Abre la vista para editar las distancias de los recorridos.

Método getRecorridos()

Este método devuelve la lista observable de recorridos cargados.

Método initialize(URL location, ResourceBundle resources)

Este método se llama automáticamente después de que se haya cargado el archivo FXML y se utiliza para inicializar el controlador. En este método se inicializa la

lista de recorridos.

Uso

Esta clase se encarga de la gestión de la vista principal de la aplicación.

Maneja la carga de datos desde archivos CSV, la visualización de los recorridos en una tabla y la navegación entre diferentes vistas de la aplicación.

Clase SecundaryController

Descripción

La clase **SecundaryController** es un controlador de la interfaz gráfica de usuario correspondiente a la vista secundaria de la aplicación. Esta clase se encarga de gestionar la lógica de la GUI y manejar las interacciones del usuario con los elementos de la pantalla relacionados con la generación de viajes.

Atributos

- **viajes**: Una lista estática de objetos **Viaje** que almacena los viajes generados en la vista secundaria.
- **recorridos**: Una lista observable de objetos **Recorrido** que almacena los datos de los recorridos cargados desde el archivo CSV.
- **transportesSeleccionados**: Una lista que almacena los tipos de transporte seleccionados para evitar su repetición en el ComboBox de transporte.

Métodos

Método setViajes(List<Viaje> viajes)

Este método establece la lista de viajes en la vista secundaria.

Método initialize()

Este método se llama automáticamente después de cargar el archivo FXML y se

utiliza para inicializar el controlador. Configura el ComboBox de punto inicial y agrega un listener para actualizar el ComboBox de punto final cuando se selecciona un punto inicial.

Método setRecorridos(ObservableList<Recorrido> recorridos)

Este método establece la lista de recorridos en la vista secundaria y actualiza los ComboBox de punto inicial y punto final con los puntos disponibles.

Método GenerarButton(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Generar Viaje". Crea un nuevo viaje con los datos seleccionados por el usuario y lo agrega a la lista de viajes.

Método updateTransporteComboBox()

Este método actualiza el ComboBox de tipo de transporte para evitar la selección de tipos de transporte ya utilizados.

Método getViajes()

Este método devuelve la lista de viajes generados en la vista secundaria.

Método addViaje(Viaje viaje)

Este método añade un nuevo viaje a la lista de viajes en la vista secundaria.

Uso

- Esta clase se encarga de la gestión de la vista secundaria de la aplicación.
- Maneja la generación de nuevos viajes y la actualización de los ComboBox de puntos y tipos de transporte.

Clase EditarDistanciaController

Descripción

La clase EditarDistanciaController es un controlador de la interfaz gráfica de usuario correspondiente a la ventana de edición de distancias de recorridos. Esta clase se encarga de gestionar la lógica de la GUI y manejar las interacciones del usuario con los elementos de la pantalla relacionados con la edición de distancias.

Atributos

idField: Un campo de texto (TextField) que permite al usuario ingresar el ID del recorrido que desea editar.

distanciaField: Un campo de texto (TextField) que permite al usuario ingresar la nueva distancia para el recorrido.

recorridosController: Una referencia al controlador principal (PrimaryController) para acceder a la lista de recorridos y actualizar la tabla después de realizar cambios.

Métodos

Método setRecorridosController(PrimaryController recorridosController)

Este método establece el controlador principal para poder acceder a la lista de recorridos desde el controlador de edición de distancias.

Método guardarCambios(ActionEvent event)

Este método se ejecuta cuando se hace clic en el botón "Guardar Cambios". Obtiene el ID y la nueva distancia ingresados por el usuario, busca el recorrido correspondiente en la lista de recorridos y actualiza su distancia. Luego, actualiza la tabla en la interfaz gráfica mediante el método actualizarTabla() del controlador

principal. Finalmente, cierra la ventana de edición.

Uso

Esta clase se encarga de la gestión de la ventana de edición de distancias de recorridos.

Permite al usuario ingresar el ID y la nueva distancia para un recorrido y guardar los cambios.

Clase ViajeController

Descripción

La clase ViajeController es un controlador de la interfaz gráfica de usuario correspondiente a la gestión de viajes. Esta clase se encarga de gestionar la lógica de la GUI y manejar las interacciones del usuario con los elementos de la pantalla relacionados con los viajes, incluida la animación de los vehículos en el mapa y la actualización de información relacionada con los viajes.

Atributos

viajes: Una lista de objetos Viaje que representa los viajes disponibles.

imageView1, imageView2, imageView3: Objetos ImageView que representan las imágenes de los vehículos para los viajes respectivos.

Otros atributos incluyen botones, etiquetas (Label), cajas de contenedores (HBox), etc., que representan los elementos de la interfaz gráfica de usuario.

Métodos

Métodos IniciarViaje1Button, IniciarViaje2Button, IniciarViaje3Button

Estos métodos se llaman cuando el usuario hace clic en los botones de inicio de viaje para los viajes 1, 2 y 3 respectivamente. Llaman al método moverImagen() para iniciar la animación del vehículo correspondiente.

Métodos Recargar1Button, Recargar2Button, Recargar3Button

Estos métodos se llaman cuando el usuario hace clic en los botones de recarga de gasolina para los viajes 1, 2 y 3 respectivamente. Llaman al método moverImagen() para continuar la animación del vehículo correspondiente y actualizar la cantidad de gasolina consumida.

Métodos Regresar1, Regresar2, Regresar3

Estos métodos se llaman cuando el usuario hace clic en los botones de regreso para los viajes 1, 2 y 3 respectivamente. Llaman al método moverImagen() para regresar el vehículo a su posición inicial.

Método IniciarTodolosViajesButton

Este método se llama cuando el usuario hace clic en el botón para iniciar todos los viajes simultáneamente. Llama al método moverImagen() para iniciar la animación de todos los vehículos al mismo tiempo.

Método setViajes

Este método establece la lista de viajes y configura la interfaz gráfica de usuario en consecuencia, mostrando la información de cada viaje y asignando imágenes a los vehículos.

Otros métodos

actualizarPuntosIniciales(), actualizarPuntosFinales(), actualizarTransporte(): Estos métodos actualizan la información mostrada en la GUI sobre los puntos inicial y final, así como el tipo de transporte para cada viaje.

actualizarDistancia(): Este método busca y actualiza la distancia de cada viaje basándose en los puntos inicial y final, utilizando la información de los recorridos del controlador principal.

moverImagen(): Este método se encarga del movimiento de un vehículo (imagen) en la interfaz gráfica de usuario desde su posición actual hasta una posición dada, simulando el progreso del viaje. También maneja la lógica relacionada con la recarga de gasolina durante el viaje.

Uso

Esta clase se encarga de la gestión de la interfaz gráfica de usuario relacionada con los viajes.

Permite al usuario iniciar, recargar y regresar los viajes, mostrando información relevante sobre los viajes en la pantalla y animando el movimiento de los vehículos en el mapa.

Clase Viaje

La clase **Viaje** representa un viaje en la aplicación. Contiene información sobre el punto inicial, el punto final, el tipo de transporte, la capacidad de gasolina, el consumo de gasolina y el recorrido total.

Campos:

- **puntoInicial**: String - El punto de inicio del viaje.
- **puntoFinal**: String - El punto de destino del viaje.
- **transporte**: String - El tipo de transporte utilizado para el viaje.

- **capacidadGasolina**: double - La capacidad de gasolina del vehículo utilizado para el viaje.
- **consumoGasolina**: double - El consumo de gasolina por unidad de distancia del vehículo.
- **recorridoTotal**: double - La distancia total recorrida en el viaje.

Métodos:

- **getPuntoInicial()**: String - Retorna el punto inicial del viaje.
- **setPuntoInicial(String puntoInicial)**: void - Establece el punto inicial del viaje.
- **getPuntoFinal()**: String - Retorna el punto final del viaje.
- **setPuntoFinal(String puntoFinal)**: void - Establece el punto final del viaje.
- **getTransporte()**: String - Retorna el tipo de transporte utilizado en el viaje.
- **setTransporte(String transporte)**: void - Establece el tipo de transporte del viaje.
- **getCapacidadGasolina()**: double - Retorna la capacidad de gasolina del vehículo.
- **setCapacidadGasolina(double capacidadGasolina)**: void - Establece la capacidad de gasolina del vehículo.
- **getConsumoGasolina()**: double - Retorna el consumo de gasolina del vehículo.
- **setConsumoGasolina(double consumoGasolina)**: void - Establece el consumo de gasolina del vehículo.
- **getRecorridoTotal()**: double - Retorna el recorrido total del viaje.
- **incrementarRecorridoTotal(double incremento)**: void - Incrementa el

recorrido total del viaje.

- **toString()**: String - Retorna una representación de cadena del objeto **Viaje**.

Clase Recorrido

La clase **Recorrido** representa un recorrido en la aplicación. Contiene información sobre el identificador del recorrido, el punto de inicio, el punto final y la distancia del recorrido.

Campos:

- **id**: SimpleStringProperty - El identificador del recorrido.
- **inicio**: SimpleStringProperty - El punto de inicio del recorrido.
- **fin**: SimpleStringProperty - El punto final del recorrido.
- **distancia**: SimpleStringProperty - La distancia del recorrido.

Constructor:

- **Recorrido(String id, String inicio, String fin, String distancia)**: Crea una instancia de **Recorrido** con el identificador, punto de inicio, punto final y distancia especificados.

Métodos:

- **getId()**: String - Retorna el identificador del recorrido.
- **setId(String id)**: void - Establece el identificador del recorrido.
- **getInicio()**: String - Retorna el punto de inicio del recorrido.
- **setInicio(String inicio)**: void - Establece el punto de inicio del recorrido.
- **getFin()**: String - Retorna el punto final del recorrido.
- **setFin(String fin)**: void - Establece el punto final del recorrido.
- **getDistancia()**: String - Retorna la distancia del recorrido.

- **setDistancia(String distancia)**: void - Establece la distancia del recorrido.

LIBRERÍAS

javafx.fxml.FXML: Esta anotación se utiliza para marcar los campos, métodos o tipos de un controlador de FXML (XML de la Interfaz de Usuario de JavaFX) para que el cargador de FXML pueda inyectar valores en ellos durante el proceso de carga del archivo FXML.

javafx.fxml.FXMLLoader: Esta clase se utiliza para cargar archivos FXML que definen la interfaz de usuario de JavaFX. Carga el archivo FXML y crea una jerarquía de nodos de la interfaz de usuario que pueden ser utilizados en la aplicación.

javafx.scene.control.Alert: Esta clase representa un cuadro de diálogo de alerta que se puede mostrar para informar al usuario sobre eventos importantes o mensajes de error.

javafx.scene.control.Button: Esta clase representa un botón en la interfaz de usuario de JavaFX. Los botones pueden tener texto o iconos y se pueden configurar para responder a eventos de clic.

javafx.scene.control.cell.PropertyValueFactory: Esta clase se utiliza para asociar las propiedades de un objeto Java a las columnas de una tabla en una vista de tabla de JavaFX. Ayuda a definir cómo se deben mostrar los datos en cada columna.

javafx.scene.Parent: Esta clase representa el nodo raíz de la jerarquía de nodos

de la interfaz de usuario de JavaFX. Se utiliza como contenedor para otros nodos de la interfaz de usuario.

javafx.scene.Scene: Esta clase representa una escena en una aplicación de JavaFX. La escena contiene todos los nodos de la interfaz de usuario que se mostrarán en la ventana principal de la aplicación.

javafx.stage.Stage: Esta clase representa la ventana principal de una aplicación de JavaFX. Es el contenedor principal para todas las escenas de la aplicación.

javafx.collections.FXCollections: Esta clase proporciona métodos estáticos para crear instancias de implementaciones de la interfaz ObservableList. Se utiliza para crear listas observables que pueden ser utilizadas para almacenar y gestionar datos en JavaFX.

javafx.collections.ObservableList: Esta interfaz representa una lista que permite a los desarrolladores registrar observadores que serán notificados cuando la lista cambie.

javafx.event.ActionEvent: Esta clase representa un evento de acción que se genera cuando se produce una acción, como hacer clic en un botón.

java.net.URL: Esta clase representa una URL (Localizador de Recursos Uniforme). Se utiliza para representar la ubicación de un recurso en Internet o en el sistema de archivos local.

java.nio.file.Paths: Esta clase proporciona métodos estáticos para convertir cadenas de rutas en objetos Path.

java.util.ArrayList: Esta clase implementa una lista de tamaño variable en Java. Se utiliza para almacenar una colección de elementos que pueden crecer o reducirse según sea necesario.

java.util.List: Esta interfaz representa una colección ordenada de elementos. Se utiliza para almacenar una secuencia de elementos en Java.

java.util.ResourceBundle: Esta clase se utiliza para cargar archivos de propiedades que contienen información localizada.

java.io.File: Esta clase representa un archivo o directorio en el sistema de archivos. Se utiliza para interactuar con archivos y directorios en el sistema de archivos local.

java.io.FileReader: Esta clase se utiliza para leer caracteres de un archivo de texto.

java.io.FileWriter: Esta clase se utiliza para escribir caracteres en un archivo de texto.

java.io.IOException: Esta clase representa una excepción de entrada/salida que

se produce cuando se produce un error durante la lectura o escritura de datos.

java.nio.file.Path: Esta interfaz representa una secuencia de nodos en un sistema de archivos. Se utiliza para representar una ruta de acceso a un archivo o directorio en el sistema de archivos local.

com.opencsv.CSVReader: Esta clase se utiliza para leer datos de archivos CSV (valores separados por comas).

com.opencsv.CSVWriter: Esta clase se utiliza para escribir datos en archivos CSV.

com.opencsv.exceptions.CsvValidationException: Esta clase representa una excepción que se produce durante la validación de datos en un archivo CSV.

javafx.scene.control.TableView: Esta clase representa una vista de tabla en la interfaz de usuario de JavaFX. Se utiliza para mostrar datos en forma de filas y columnas.

javafx.scene.control.TableColumn: Esta clase representa una columna en una vista de tabla en la interfaz de usuario de JavaFX. Se utiliza para definir las columnas y configurar cómo se deben mostrar los datos en ellas.

javafx.stage.FileChooser: Esta clase se utiliza para mostrar un cuadro de diálogo que permite al usuario seleccionar archivos del sistema de archivos local. Se utiliza comúnmente para cargar o guardar archivos en una aplicación.