INFORMACION DE CADA CLASE:

- ControladorRutas

Encontrar Rutas: La función encontrarRutas utiliza un algoritmo de búsqueda en profundidad (DFS) para encontrar todas las posibles rutas entre un origen y un destino dados, utilizando una lista de conexiones de grafo proporcionada.
Llenar Tablas de Origen y Destino: Las funciones llenarTablasOrigen y llenarTablasDestino se utilizan para llenar tablas de origen y destino respectivamente en una interfaz gráfica de usuario (GUI), extrayendo los datos de la lista de conexiones de grafo proporcionada.
Nombre de Ruta: La función nombreRuta devuelve el nombre de la ruta seleccionada en una tabla de origen o destino en la GUI.
Llenar Combo de Rutas: La función llenarComboRutas llena un componente de selección (como un JComboBox) con nombres de rutas para que el usuario pueda elegir entre ellas.
Eliminar Contenido de Carpeta: La función eliminarContenidoCarpeta elimina todo el contenido de una carpeta especificada en el sistema de archivos.
Información de Ruta: La función informacionDeRuta calcula y muestra información detallada sobre una ruta seleccionada, como la distancia total, el consumo de gas (si es un vehículo), la velocidad promedio, etc.
Mejor Ruta: La función mejorRuta determina la ruta más eficiente entre un conjunto de rutas, teniendo en cuenta diferentes criterios como la distancia total o la eficiencia (dependiendo de si es un vehículo o a pie).
Graficar Grafo Completo: La función graficarGrafoCompleto genera una representación gráfica del grafo completo de conexiones entre ubicaciones, posiblemente para visualización o análisis adicional.
En resumen, esta clase proporciona funcionalidades para gestionar y analizar rutas entre ubicaciones, así como para mostrar información relevante sobre estas rutas en una interfaz de usuario.

- ControladorVista2 parte del sistema que gestiona y visualiza rutas entre ubicaciones. Funciones principales:

Llenar Tabla de Posiciones o Lugares: Esta función llena una tabla con los lugares involucrados en una lista de rutas, mostrando tanto los destinos como los puntos de origen de las rutas.
Llenar Combo con Rutas: Llena un componente de selección con nombres de rutas para que el usuario pueda elegir entre ellas.
Llenar Combo Funcionalidades: Llena un componente de selección con diferentes opciones de funcionalidades relacionadas con la selección de rutas, como la mejor ruta en base a criterios específicos.
Generar Árboles B de Imágenes: Esta función genera árboles B a partir de las listas de rutas, utilizando diferentes criterios (como consumo de gasolina, desgaste físico, distancia, etc.) y genera imágenes de las rutas más eficientes y sus respectivos árboles B.
Probabilidad de Tráfico: Calcula la probabilidad de tráfico en función del horario y el grafo proporcionado.
Devolver Valor Menor y Mayor: Estas funciones devuelven el índice del valor menor y mayor respectivamente en una lista de hojas.
Traer Velocidad: Calcula la velocidad en función de si es un vehículo o se está caminando, tomando en cuenta el tiempo y la probabilidad de tráfico.

- **Datos** es una clase de almacenamiento de datos que contiene listas estáticas de objetos Grafo y Horario.

Almacenamiento de Datos: La clase Datos proporciona un lugar centralizado para almacenar listas estáticas de objetos Grafo y Horario, que probablemente contienen información sobre conexiones entre lugares y horarios asociados a esas conexiones.

- La clase **main** es la clase principal de la aplicación. Lo que hace:

Método Main: El método main es el punto de entrada de la aplicación. En este método se instancia un objeto de la clase cargaDatos, que probablemente es una ventana de carga de datos.
Configuración de la Ventana: Se realizan algunas configuraciones en la ventana de carga de datos recién creada, como hacerla visible, habilitarla, deshabilitar la posibilidad de redimensionarla y centrarla en la pantalla.

- **Grafo** representa un nodo en un grafo, donde cada nodo tiene los siguientes atributos:

origen: Una cadena que representa el punto de origen del grafo.
destino: Una cadena que representa el punto de destino del grafo.
tiempo_vehiculo: Un entero que indica el tiempo necesario para viajar de origen a destino en un vehículo.
tiempo_pie: Un entero que indica el tiempo necesario para viajar de origen a destino a pie.
consumo_gas: Un entero que representa el consumo de gasolina en el viaje de origen a destino.
desgaste_personal: Un entero que indica el desgaste personal en el viaje de origen a destino.
distancia: Un entero que representa la distancia entre el origen y el destino.
Además de los atributos, la clase proporciona métodos para acceder y modificar estos atributos, así como constructores para crear instancias de la clase.

- **Horario** representa un horario específico para un tramo de viaje entre dos puntos en el modelo. Los atributos de esta clase son:

origen: Una cadena que representa el punto de origen del horario.
destino: Una cadena que representa el punto de destino del horario.
hora_inicio: Un entero que indica la hora de inicio del horario.
hora_finalizada: Un entero que indica la hora de finalización del horario.
probabilidad_trafico: Un entero que representa la probabilidad de tráfico en ese horario.
La clase proporciona métodos para acceder y modificar estos atributos, así como constructores para crear instancias de la clase.

- **ArbolB** implementa un árbol B genérico.

Atributos:
M: Define el grado máximo del árbol B.
lista: Una lista de listas de objetos Grafo, utilizada para graficar.
valorTabla: Una lista de objetos Hoja, utilizada para mantener los valores y sus índices en el árbol.
raiz: El nodo raíz del árbol.
Constructor:
Crea un árbol B inicializando la raíz como una hoja.
Métodos:
insertar(valor, indiceTabla, valorN): Inserta un valor en el árbol, manteniendo su índice y valor en la tabla de valores.

imprimir(): Imprime el árbol (recorrido inorden).
graficar(archivoDot, archivoImagen, lista): Genera una representación gráfica del árbol utilizando Graphviz.
graficarNodo(nodo, out, lista): Método auxiliar recursivo para generar la definición de nodos y enlaces para la graficación.
devolverIndiceTabla(valor, nodo): Devuelve el índice correspondiente en la tabla de valores dado un valor y un nodo.
valorGraficar(lista): Devuelve un valor formateado para la graficación, concatenando los valores de los objetos Grafo en la lista.
Esta clase permite la inserción de elementos, impresión y graficación del árbol B.

- **Graficar** proporciona métodos para generar gráficos de diferentes tipos de rutas y grafos.

Métodos:
graficarNuevoGrafoCompleto(lista, ruta): Genera un gráfico de un nuevo grafo completo basado en una lista de listas de objetos Grafo. Guarda el gráfico en el directorio especificado por ruta.
graficarGrafoCompleto(lista, carpeta, nombre): Genera un gráfico de un grafo completo basado en una lista de objetos Grafo. Guarda el gráfico en el directorio especificado por carpeta con el nombre dado por nombre.
graficarRutaPorRuta(listaRutas, index, contadorImagenes, carpeta, nombre): Genera un gráfico de ruta por ruta basado en una lista de objetos Grafo que representan una ruta específica. Guarda el gráfico en el directorio especificado por carpeta con el nombre dado por nombre, junto con el índice de la ruta y un contador de imágenes.
Estos métodos utilizan la biblioteca Graphviz para generar los gráficos en formato PNG. Cada método recorre la lista de objetos Grafo y agrega nodos y conexiones al gráfico, luego renderiza y guarda el gráfico en un archivo PNG. Además, hay un método main que permite al usuario ingresar pares de nodos para generar un gráfico de un grafo completo y guardarlo como arreglo.png.

- **Hoja** representa un nodo hoja que se utiliza en la implementación de un árbol B.

Atributos:
llave: Almacena el valor con el que se va a indexar cada hoja en el árbol.
valor: Representa el identificador del ArrayList.
Métodos:
Hoja(llave, valor): Constructor que inicializa una hoja con la llave y el valor especificados.
getLlave(): Método para obtener la llave de la hoja.
setLlave(llave): Método para establecer la llave de la hoja.
getValor(): Método para obtener el valor de la hoja.
setValor(valor): Método para establecer el valor de la hoja.
Esta clase proporciona una estructura básica para almacenar información en nodos hoja en el contexto de un árbol B.

- **Lector** se encarga de leer datos desde un archivo de texto y agregarlos a las listas de grafos o horarios según se especifique.

Atributos:
texto: Almacena el contenido del archivo de texto.
linea: Almacena cada línea leída del archivo de texto.
bufer: Objeto BufferedReader utilizado para leer el archivo de texto.

Métodos:
leer(nombreArchivo, esGrafo): Lee el contenido del archivo especificado por nombreArchivo y, según el valor de esGrafo, agrega los datos a la lista de grafos o de horarios.
agregarGrafoADatos(linea): Analiza la línea proporcionada, la divide en partes y crea un nuevo objeto Grafo, luego lo agrega a la lista de grafos en la clase Datos.
agregarHorariosADatos(linea): Similar a agregarGrafoADatos, pero para objetos Horario.
La clase proporciona una forma de cargar datos desde un archivo de texto estructurado en el formato correcto y agregarlos a las listas correspondientes en la clase Datos.

- **Reloj** es una utilidad que muestra la hora actual y la actualiza automáticamente cada segundo.

Atributos:
labelHora: JLabel donde se muestra la hora.
timer: Objeto Timer que actualiza la hora automáticamente.
horaModificate, minuteModificate, secondModifcate: Variables para almacenar la hora modificada.
Constructor:
Reloj(JLabel labelHora): Inicializa el reloj con la etiqueta especificada y comienza la actualización automática de la hora.
Métodos:
iniciarActualizacionAutomatica(boolean editado): Inicia la actualización automática del reloj.
actualizarHora(boolean editado): Actualiza la hora mostrada en la etiqueta.
modificarHora(String nuevaHora): Modifica la hora mostrada en el reloj.
iniciarActualizacionAutomaticaHoraEdit(): Inicia la actualización automática del reloj después de una modificación.
modificarTiempos(int h, int m, int s): Modifica los tiempos de la hora modificada y actualiza la etiqueta de la hora.
La clase permite mostrar la hora actual y modificarla según sea necesario, manteniendo la actualización automática en ambos casos.

- **cargaDatos** es una ventana de la interfaz gráfica de usuario (GUI) que permite al usuario cargar archivos de datos, ya sea para grafos o para horarios.

Atributos:
rutasController: Controlador de rutas asociado a la clase.
Métodos:
initComponents(): Inicializa los componentes de la interfaz gráfica.
jButton1ActionPerformed(java.awt.event.ActionEvent evt): Maneja el evento de clic en el botón "Cargar Lugares(Grafos)", abre un JFileChooser para seleccionar un archivo y llama al método abrirFileChooser(true) para cargar los datos.
abrirFileChooser(boolean esGrafo): Abre un JFileChooser para seleccionar un archivo y llama al método leer de la clase Lector para cargar los datos.
abrirVentanaPrincipal(): Abre la ventana principal de la aplicación y oculta la ventana actual.
jButton2ActionPerformed(java.awt.event.ActionEvent evt): Maneja el evento de clic en el botón "Ahora no", abre la ventana principal de la aplicación y oculta la ventana actual.
jButton3ActionPerformed(java.awt.event.ActionEvent evt): Maneja el evento de clic en el botón "Cargar Horarios", abre un JFileChooser para seleccionar un archivo y llama al método abrirFileChooser(false) para cargar los datos.
La clase proporciona una interfaz simple para que el usuario cargue archivos de datos y los procese en la aplicación.

- **cargaDatos2** es otra ventana de la interfaz gráfica de usuario (GUI) que permite al usuario cargar archivos de datos, ya sea para lugares, rutas o grafos, y para horarios de rutas.

Métodos:
initComponents(): Inicializa los componentes de la interfaz gráfica.
jButton1ActionPerformed(java.awt.event.ActionEvent evt): Maneja el evento de clic en el botón "Lugares, rutas o grafos", abre un JFileChooser para seleccionar un archivo y llama al método abrirFileChooser(true) para cargar los datos.
abrirFileChooser(boolean esGrafo): Abre un JFileChooser para seleccionar un archivo y llama al método leer de la clase Lector para cargar los datos.
jButton2ActionPerformed(java.awt.event.ActionEvent evt): Maneja el evento de clic en el botón "Horarios de rutas", abre un JFileChooser para seleccionar un archivo y llama al método abrirFileChooser(false) para cargar los datos.
La clase proporciona una interfaz simple para que el usuario cargue archivos de datos relacionados con lugares, rutas, grafos y horarios de rutas en la aplicación.

- **vista** representa la interfaz principal de la aplicación y contiene varios componentes y métodos para interactuar con los datos y la interfaz gráfica.

Atributos:
rutasController: Objeto de la clase ControladorRutas utilizado para controlar las rutas.
contadorImagenes: Contador utilizado para mantener un registro de las imágenes generadas.
listaContadorImages: Lista utilizada para almacenar los contadores de imágenes.
rutas: ArrayList utilizado para almacenar las rutas.
reloj: Objeto de la clase Reloj utilizado para mostrar la hora en la interfaz.
Métodos:
initOthersComponents(JLabel label, JPanel panel): Método privado para inicializar los componentes de la interfaz gráfica.
llamarImagenGrafoCompleto(): Método para cargar la imagen del grafo completo en la interfaz.
llamarImagenDeCadaRuta(String nameImg): Método para cargar la imagen de cada ruta en la interfaz.
btnVerRutasActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de clic en el botón "Ver Rutas". Encuentra y muestra las rutas disponibles.
comboRutasActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de selección en el combo de rutas. Muestra la información de la ruta seleccionada.
jButton1ActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de clic en el botón "Seleccionar Ruta". Abre una nueva ventana para mostrar detalles adicionales de la ruta seleccionada.
jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de clic en el menú "Cargar datos". Abre una ventana para cargar datos desde archivos.
jButton2ActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de clic en el botón "Editar". Permite al usuario modificar la hora del reloj.
jButton3ActionPerformed(java.awt.event.ActionEvent evt): Método que maneja el evento de clic en el botón "Iniciar Viaje". Crea un grafo dirigido y actualiza la interfaz con los datos de las rutas disponibles.
La clase vista proporciona una interfaz interactiva para que el usuario interactúe con los datos de las rutas y la aplicación en general.

- **vista2** A continuación, se presentan algunos puntos clave:

Variables y atributos:
La clase tiene varios atributos, como rutasController, vista, esVehiculo, destino, origen, listarutas, nuevasRutas, controladorRutas2, reloj, contadorImagenes, listaContadorImages, contadorImagesFuncionalidades y listaImagesFUncionalidades.
Estos atributos se utilizan para almacenar información relevante para el funcionamiento del programa.

Métodos y funcionalidades:
La clase contiene varios métodos, como initOthersComponents, llamarImagenALabelGrafo, y otros.
Estos métodos se utilizan para inicializar componentes gráficos, cargar imágenes y realizar otras tareas relacionadas con la interfaz de usuario y la lógica del programa.

Interfaz gráfica:
La clase crea una ventana de interfaz gráfica (JFrame) con varios paneles (JPanel) y etiquetas (JLabel).
Se utilizan componentes como JScrollPane para mostrar imágenes y permitir desplazamiento.

Funcionalidad principal:
Parece que la clase está diseñada para mostrar rutas y gráficos relacionados con estas rutas.
Hay referencias a tablas (jTable1), etiquetas de posición actual (labelPosicionActual), y selección de funcionalidades (comboFuncionalidades).

En resumen, la clase vista2 forma parte de una aplicación que visualiza rutas y gráficos, y proporciona funcionalidades relacionadas con la gestión de rutas.

- **vistaArbol** es una ventana de visualización de un árbol B generado a partir de una ruta de archivo.

Constructor:
Recibe una ruta como parámetro.
Inicializa los componentes de la ventana.
Llama al método llamarImagenALabelGrafo() para cargar la imagen del árbol B.
Método llamarImagenALabelGrafo(String ruta):
Carga una imagen desde la ruta especificada.
Crea un ImageIcon con la imagen cargada.
Asigna el ImageIcon al JLabel llamado labelArbolB.
Método initOthersComponents(JLabel label, JPanel panel):
Inicializa los componentes adicionales de la ventana.
Crea un JPanel que contiene al JLabel con la imagen.
Crea un JScrollPane y agrega el JPanel a él.
Establece las propiedades del JScrollPane.
Establece el tamaño preferido del JScrollPane.
Agrega el JScrollPane al JPanel pasado como argumento.
La ventana tiene un JLabel llamado labelArbolB donde se muestra la imagen del árbol B. La imagen se coloca dentro de un JScrollPane para permitir el desplazamiento si la imagen es más grande que el área visible.

En resumen, esta clase proporciona una interfaz gráfica simple para visualizar un árbol B generado desde un archivo en una ruta especificada