



30 DE JUNIO DE 2020

MANUAL TECNICO

PROYECTO 2 “LLEGA RAPIDITO”

ARACELY JACQUELINE MÉNDEZ GONZÁLEZ 201800491

DIDIER ALFREDO DOMÍNGUEZ URÍAS 201801266

HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

Para la realización del proyecto se utilizó el lenguaje de programación Java en su versión de JDK 8 mediante la utilización del IDE Apache NetBeans 11.3, para el desarrollo de los reportes se utilizó la herramienta Graphviz en su versión 2.44

ESTRUCTURA

RUTAS (LISTA DE ADYACENCIA)

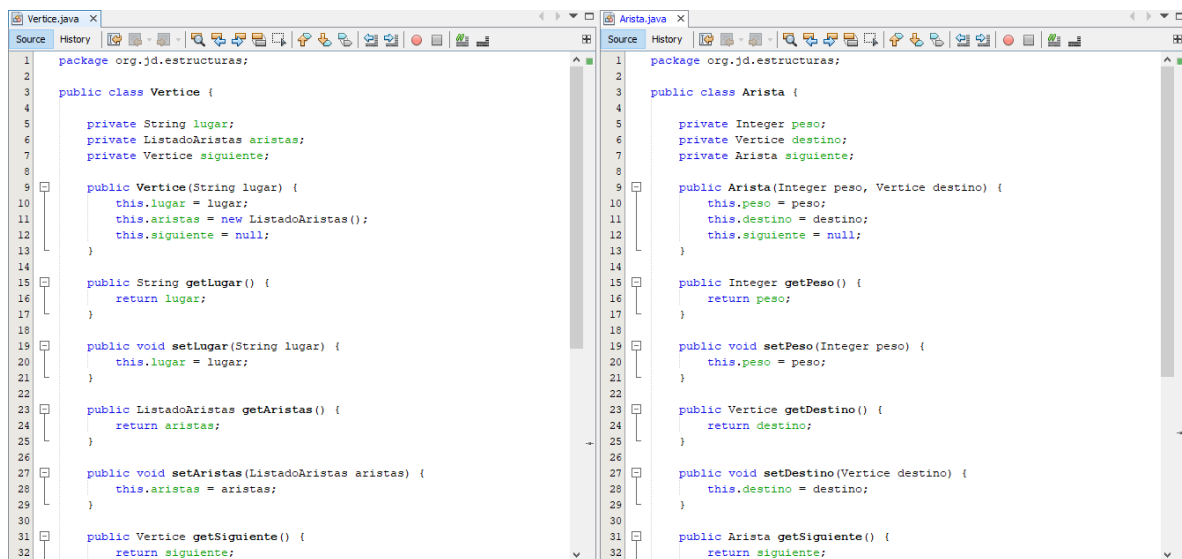
Para el manejo de las rutas se implementó un grafo dirigido el cual se obtiene por medio de una lista de adyacencia la cual se encuentra estructurada por medio de la codificación de dos listas simples la primera que sirve para almacenar cada uno de los lugares (vértices) y la segunda para realizar cada una de las conexiones a los lugares destino (aristas) esta lista se ordena por medio de los tiempos de llegada a cada uno (pesos).

Vértices

- Nombre del lugar
- Lugares destino (aristas)
- Puntero al vértice siguiente

Aristas

- Tiempo (peso)
- Destino (vértice)
- Puntero a la arista siguiente



```
package org.jd.estructuras;

public class Vertice {
    private String lugar;
    private ListadoAristas aristas;
    private Vertice siguiente;

    public Vertice(String lugar) {
        this.lugar = lugar;
        this.aristas = new ListadoAristas();
        this.siguiente = null;
    }

    public String getLugar() {
        return lugar;
    }

    public void setLugar(String lugar) {
        this.lugar = lugar;
    }

    public ListadoAristas getAristas() {
        return aristas;
    }

    public void setAristas(ListadoAristas aristas) {
        this.aristas = aristas;
    }

    public Vertice getSiguiente() {
        return siguiente;
    }
}

package org.jd.estructuras;

public class Arista {
    private Integer peso;
    private Vertice destino;
    private Arista siguiente;

    public Arista(Integer peso, Vertice destino) {
        this.peso = peso;
        this.destino = destino;
        this.siguiente = null;
    }

    public Integer getPeso() {
        return peso;
    }

    public void setPeso(Integer peso) {
        this.peso = peso;
    }

    public Vertice getDestino() {
        return destino;
    }

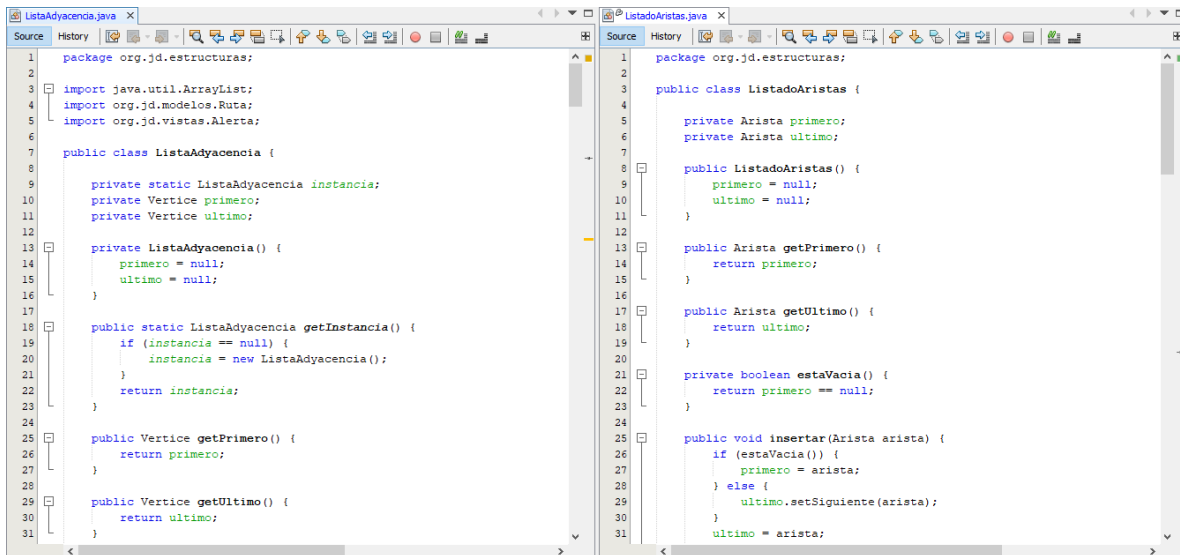
    public void setDestino(Vertice destino) {
        this.destino = destino;
    }

    public Arista getSiguiente() {
        return siguiente;
    }
}
```

Cada una de las listas posee una codificación de los métodos y funciones necesarias para realizar cada una de las operaciones básicas como lo son:

- Insertar
- Modificar
- Eliminar
- Graficar
- Ordenar (solo en la lista de aristas)

La lista principal para la lista de adyacencia es la lista de vértices la cual es necesaria para poder acceder a cada uno de los nodos y sus destinos finales.



The image shows two side-by-side windows of a Java IDE. The left window displays the source code for `ListaAdyacencia.java`, and the right window displays the source code for `ListadoAristas.java`. Both files are part of the `org.jd.estructuras` package.

```
package org.jd.estructuras;

import java.util.ArrayList;
import org.jd.modelos.Ruta;
import org.jd.vistas.Alertas;

public class ListaAdyacencia {

    private static ListaAdyacencia instancia;
    private Vertice primero;
    private Vertice ultimo;

    public ListaAdyacencia() {
        primero = null;
        ultimo = null;
    }

    public static ListaAdyacencia getInstancia() {
        if (instancia == null) {
            instancia = new ListaAdyacencia();
        }
        return instancia;
    }

    public Vertice getPrimero() {
        return primero;
    }

    public Vertice getUltimo() {
        return ultimo;
    }
}
```

```
package org.jd.estructuras;

public class ListadoAristas {

    private Arista primero;
    private Arista ultimo;

    public ListadoAristas() {
        primero = null;
        ultimo = null;
    }

    public Arista getPrimero() {
        return primero;
    }

    public Arista getUltimo() {
        return ultimo;
    }

    private boolean estaVacia() {
        return primero == null;
    }

    public void insertar(Arista arista) {
        if (estaVacia()) {
            primero = arista;
        } else {
            ultimo.setSiguiente(arista);
        }
        ultimo = arista;
    }
}
```

CLIENTES (TABLA HASH)

Para el manejo de cada uno de los clientes se implementó una tabla de hash abierta y de direccionamiento cerrado la cual posee un tamaño inicial de **$M = 37$** , el proceso de dispersión se realiza mediante una función la cual está definida como **$h(llv) = llv \bmod M$** donde h es la función de dispersión, llv es el número de DPI del cliente y M es el rango del arreglo de llaves. La tabla hash solo puede contener un máximo de 75% de ocupación en sus índices, para evitar esto es necesario aumentar la tabla en 37 cada vez que sea necesario.

Para el manejo de las colisiones se utilizó el direccionamiento cerrado el cual es manejado por medio de listas simples, cada cliente es agredido al final de cada lista simple si existe colisión en el índice a insertar.

La lista simple posee una codificación de los métodos y funciones necesarias para realizar cada una de las operaciones básicas como lo son:

- Insertar
- Modificar
- Eliminar
- Graficar

The image shows two side-by-side Java IDE windows. The left window displays the code for `NodoListaSimple.java`, which is a simple linked list node. It has a package `org.jd.estructuras`, imports `org.jd.modelos.Cliente`, and defines a `NodoListaSimple` class with private attributes `cliente` and `siguiente`. It includes methods for getting and setting these attributes and a `setSiguiente` method. The right window displays the code for `ListaSimple.java`, which is a simple linked list. It has the same package and imports, and defines a `ListaSimple` class with private attributes `primero` and `ultimo`. It includes methods for getting and setting these attributes, a `estaVacia` method, and an `agregar` method that adds a new node to the list.

```

package org.jd.estructuras;

import org.jd.modelos.Cliente;

public class NodoListaSimple {

    private Cliente cliente;
    private NodoListaSimple siguiente;

    public NodoListaSimple(Cliente cliente) {
        this.cliente = cliente;
        this.siguiente = null;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public NodoListaSimple getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoListaSimple siguiente) {
        this.siguiente = siguiente;
    }
}

```

```

package org.jd.estructuras;

import org.jd.modelos.Cliente;

public class ListaSimple {

    private NodoListaSimple primero;
    private NodoListaSimple ultimo;

    public ListaSimple() {
        primero = null;
        ultimo = null;
    }

    public NodoListaSimple getPrimero() {
        return primero;
    }

    public NodoListaSimple getUltimo() {
        return ultimo;
    }

    private boolean estaVacia() {
        return primero == null;
    }

    public void agregar(Cliente cliente) {
        NodoListaSimple nuevo = new NodoListaSimple(cliente);
        if (estaVacia()) {
            primero = nuevo;
        } else {
            ultimo.setSiguiente(nuevo);
        }
    }
}

```

Los atributos necesarios para los clientes cada uno con sus get y set respectivamente para obtener los datos y modificarlos. Los métodos necesarios para el desarrollo de la tabla hash se indican a continuación.

Cientes

- DPI
- Nombres
- Apellidos
- Genero
- Fecha de nacimiento
- Teléfono
- Dirección

Tabla Hash

- Insertar
- Buscar
- Eliminar
- Migrar Tabla
- Graficar

The image shows two side-by-side Java IDE windows. The left window displays the code for `Cliente.java`, which is a simple client model. It has a package `org.jd.modelos` and defines a `Cliente` class with private attributes `DPI`, `nombres`, `apellidos`, `genero`, `fechaNacimiento`, `telefono`, and `direccion`. It includes methods for getting and setting these attributes. The right window displays the code for `TablaHash.java`, which is a simple hash table. It has a package `org.jd.estructuras` and imports `java.util.ArrayList`, `org.jd.modelos.Cliente`, and `org.jd.vistas.Alertas`. It defines a `TablaHash` class with private static attributes `instancia`, `m`, `cantidadDatos`, and `tablaHash`. It includes methods for getting the instance, initializing the table, and adding a new client to the table.

```

package org.jd.modelos;

public class Cliente {

    private String DPI;
    private String nombres;
    private String apellidos;
    private String genero;
    private String fechaNacimiento;
    private String telefono;
    private String direccion;

    public Cliente(String DPI, String nombres, String apellidos, String genero, String fechaNacimiento, String telefono, String direccion) {
        this.DPI = DPI;
        this.nombres = nombres;
        this.apellidos = apellidos;
        this.genero = genero;
        this.fechaNacimiento = fechaNacimiento;
        this.telefono = telefono;
        this.direccion = direccion;
    }

    public String getDPI() {
        return DPI;
    }

    public void setDPI(String DPI) {
        this.DPI = DPI;
    }
}

```

```

package org.jd.estructuras;

import java.util.ArrayList;
import org.jd.modelos.Cliente;
import org.jd.vistas.Alertas;

public class TablaHash {

    private static TablaHash instancia;
    private Integer m;
    private Integer cantidadDatos;
    private ListaSimple[] tablaHash;

    private TablaHash() {
        m = 37;
        cantidadDatos = 0;
        tablaHash = new ListaSimple[m];
        inicializarTabla(tablaHash);
    }

    public static TablaHash getInstancia() {
        if (instancia == null) {
            instancia = new TablaHash();
        }
        return instancia;
    }

    private void inicializarTabla(ListaSimple[] tabla) {
        for (int i = 0; i < tabla.length; i++) {
            tabla[i] = new ListaSimple();
        }
    }
}

```

VEHÍCULOS

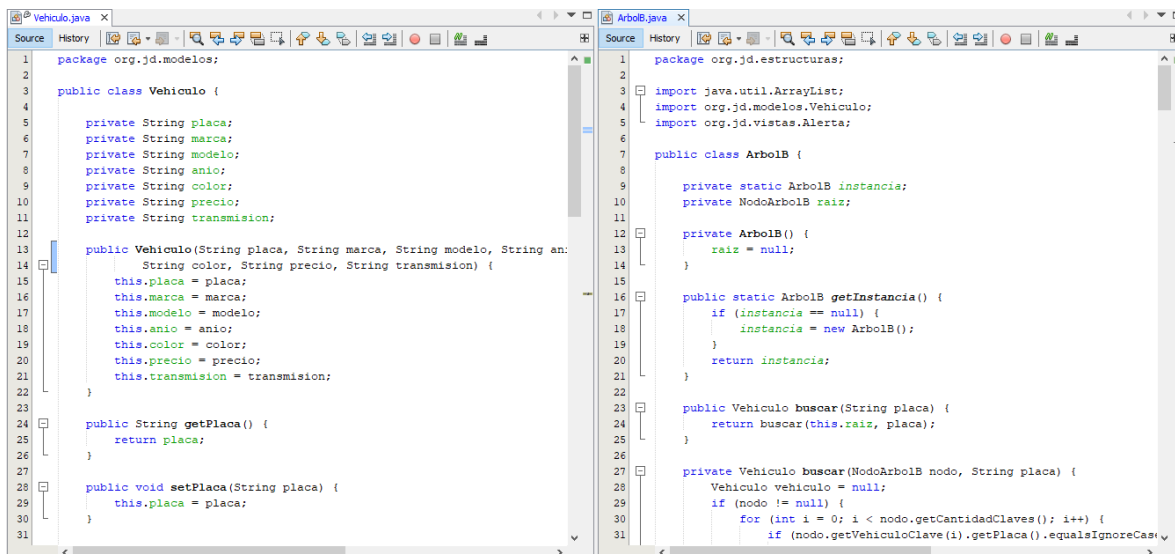
Para el manejo de cada uno de los vehículos se implementó un árbol B de orden 5, el cual se ordena y estructura por medio del número de placas de cada vehículo. Los atributos necesarios para el vehículo se describen a continuación, cada uno con sus respectivos get y set de los datos. Los métodos para el árbol B cambia dependiendo de las acciones que se van a realizar.

Vehículos

- Placa
- Marca
- Modelo
- Año
- Color
- Precio
- Transmisión

Árbol B

- Insertar
- Buscar
- Dividir
- Graficar



```
1 package org.jd.modelos;
2
3 public class Vehiculo {
4
5     private String placa;
6     private String marca;
7     private String modelo;
8     private String anio;
9     private String color;
10    private String precio;
11    private String transmision;
12
13    public Vehiculo(String placa, String marca, String modelo, String anio, String color, String precio, String transmision) {
14        this.placa = placa;
15        this.marca = marca;
16        this.modelo = modelo;
17        this.anio = anio;
18        this.color = color;
19        this.precio = precio;
20        this.transmision = transmision;
21    }
22
23    public String getPlaca() {
24        return placa;
25    }
26
27    public void setPlaca(String placa) {
28        this.placa = placa;
29    }
30
31 }
```

```
1 package org.jd.estructuras;
2
3 import java.util.ArrayList;
4 import org.jd.modelos.Vehiculo;
5 import org.jd.vistas.Alertas;
6
7 public class ArbolB {
8
9     private static ArbolB instancia;
10    private NodoArbolB raiz;
11
12    private ArbolB() {
13        raiz = null;
14    }
15
16    public static ArbolB getInstancia() {
17        if (instancia == null) {
18            instancia = new ArbolB();
19        }
20        return instancia;
21    }
22
23    public Vehiculo buscar(String placa) {
24        return buscar(this.raiz, placa);
25    }
26
27    private Vehiculo buscar(NodoArbolB nodo, String placa) {
28        Vehiculo vehiculo = null;
29        if (nodo != null) {
30            for (int i = 0; i < nodo.getCantidadClaves(); i++) {
31                if (nodo.getVehiculoClave(i).getPlaca().equalsIgnoreCase(placa)) {
```

CONDUCTORES

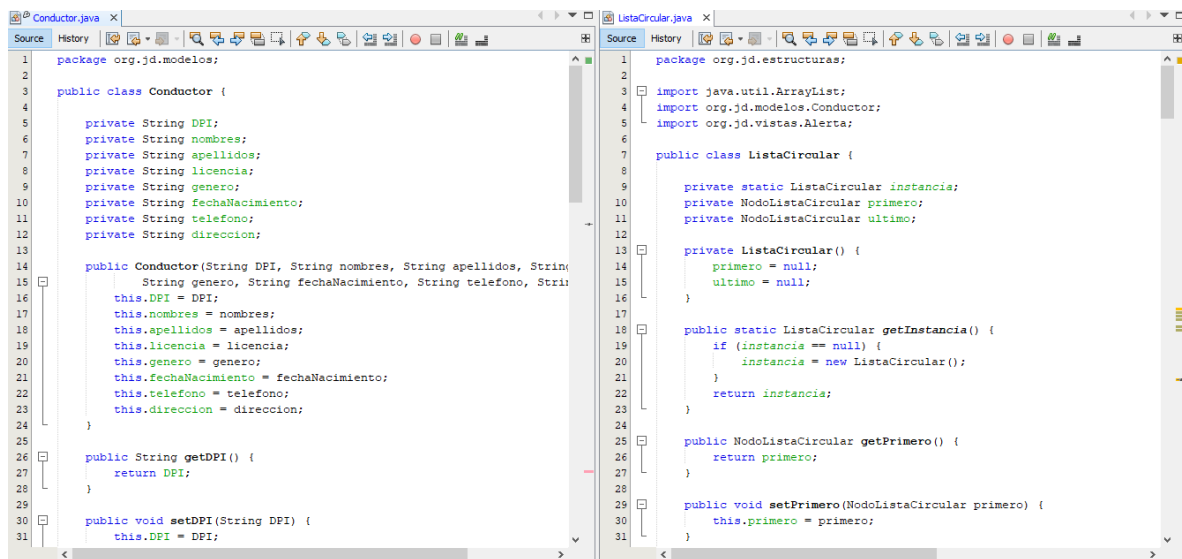
Para el manejo de cada uno de los conductores se implementó una lista circular doblemente enlazada ordenada, la cual se ordena por medio del DPI del conductor. Los atributos necesarios para el conductor son similares a la del cliente solo que este contiene el tipo de licencia, al igual cada uno con sus respectivos get y set de los datos. Los métodos para la lista circular varían dependiendo de las acciones que se van a realizar.

Conductores

- DPI
- Nombres
- Apellidos
- Licencia
- Genero
- Fecha de nacimiento
- Teléfono
- Dirección

Lista Circular Doblemente Enlazada

- Insertar
- Buscar
- Modificar
- Eliminar
- Graficar



```
1 package org.jd.modelos;
2
3 public class Conductor {
4
5     private String DPI;
6     private String nombres;
7     private String apellidos;
8     private String licencia;
9     private String genero;
10    private String fechaNacimiento;
11    private String telefono;
12    private String direccion;
13
14    public Conductor(String DPI, String nombres, String apellidos, String
15        String genero, String fechaNacimiento, String telefono, Stri
16        this.DPI = DPI;
17        this.nombres = nombres;
18        this.apellidos = apellidos;
19        this.licencia = licencia;
20        this.genero = genero;
21        this.fechaNacimiento = fechaNacimiento;
22        this.telefono = telefono;
23        this.direccion = direccion;
24    }
25
26    public String getDPI() {
27        return DPI;
28    }
29
30    public void setDPI(String DPI) {
31        this.DPI = DPI;
32    }
33
34 }
```

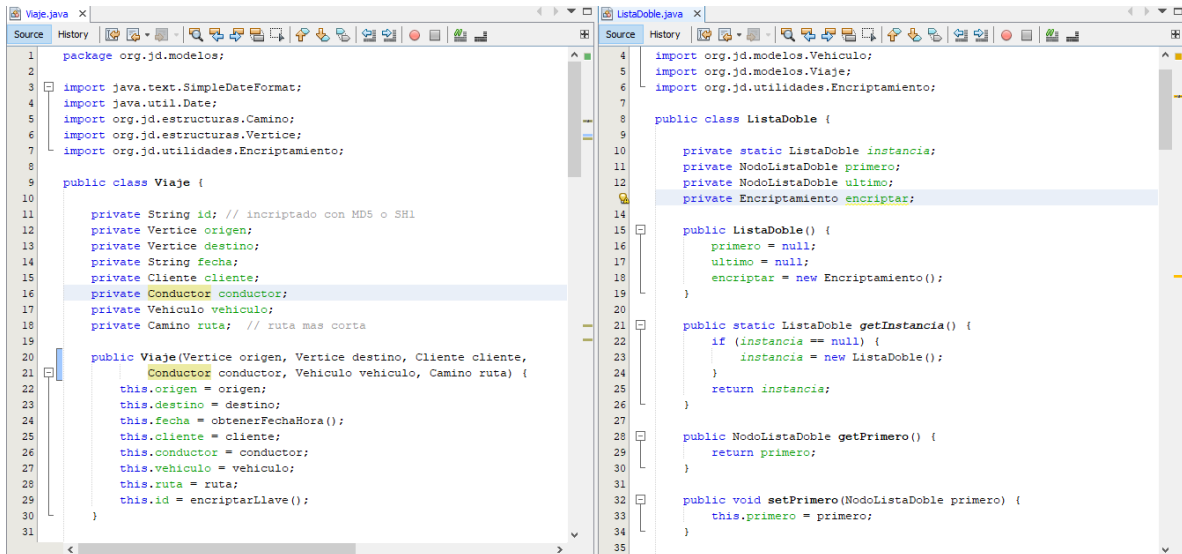
```
1 package org.jd.estructuras;
2
3 import java.util.ArrayList;
4 import org.jd.modelos.Conductor;
5 import org.jd.vistas.Alertas;
6
7 public class ListaCircular {
8
9     private static ListaCircular instancia;
10    private NodoListaCircular primero;
11    private NodoListaCircular ultimo;
12
13    private ListaCircular() {
14        primero = null;
15        ultimo = null;
16    }
17
18    public static ListaCircular getInstancia() {
19        if (instancia == null) {
20            instancia = new ListaCircular();
21        }
22        return instancia;
23    }
24
25    public NodoListaCircular getPrimero() {
26        return primero;
27    }
28
29    public void setPrimero(NodoListaCircular primero) {
30        this.primero = primero;
31    }
32
33 }
```

VIAJES

Para el manejo de cada uno de los viajes se implementó una lista doblemente enlazada la cual es ordenada por la llave generada y encriptada en el momento de crearse un nuevo viaje. Al crearse un viaje por medio del lugar de origen y el lugar de destino se calcula el camino más corto de llegada basándose en los tiempos (pesos) para recorrer de llegada a cada lugar, para calcular el camino más corto se implementó el método de Dijkstra utilizando una tabla creada por medio de listas simples y una cola para guardar los lugares visitados. Los atributos de los viajes se describen a continuación cada uno con los get y set de información, tomando en cuenta que los atributos de cliente, vehículo y conductor deben hacer referencia a los datos ya existente en las otras estructuras.

Viajes

- ID (encriptado)
- Origen
- Destino
- Fecha
- Cliente
- Conductor
- Vehículo
- Camino



The screenshot shows two IDE windows side-by-side. The left window displays the code for the `Viaje` class, and the right window displays the code for the `ListaDoble` class.

```
package org.jd.modelos;

import java.text.SimpleDateFormat;
import java.util.Date;
import org.jd.estructuras.Camino;
import org.jd.estructuras.Vertice;
import org.jd.utilidades.Encriptamiento;

public class Viaje {

    private String id; // incriptado con MD5 o SHA1
    private Vertice origen;
    private Vertice destino;
    private String fecha;
    private Cliente cliente;
    private Conductor conductor;
    private Vehiculo vehiculo;
    private Camino ruta; // ruta mas corta

    public Viaje(Vertice origen, Vertice destino, Cliente cliente,
        Conductor conductor, Vehiculo vehiculo, Camino ruta) {
        this.origen = origen;
        this.destino = destino;
        this.fecha = obtenerFechaHora();
        this.cliente = cliente;
        this.conductor = conductor;
        this.vehiculo = vehiculo;
        this.ruta = ruta;
        this.id = encriptarLlave();
    }
}
```

```
import org.jd.modelos.Vehiculo;
import org.jd.modelos.Viaje;
import org.jd.utilidades.Encriptamiento;

public class ListaDoble {

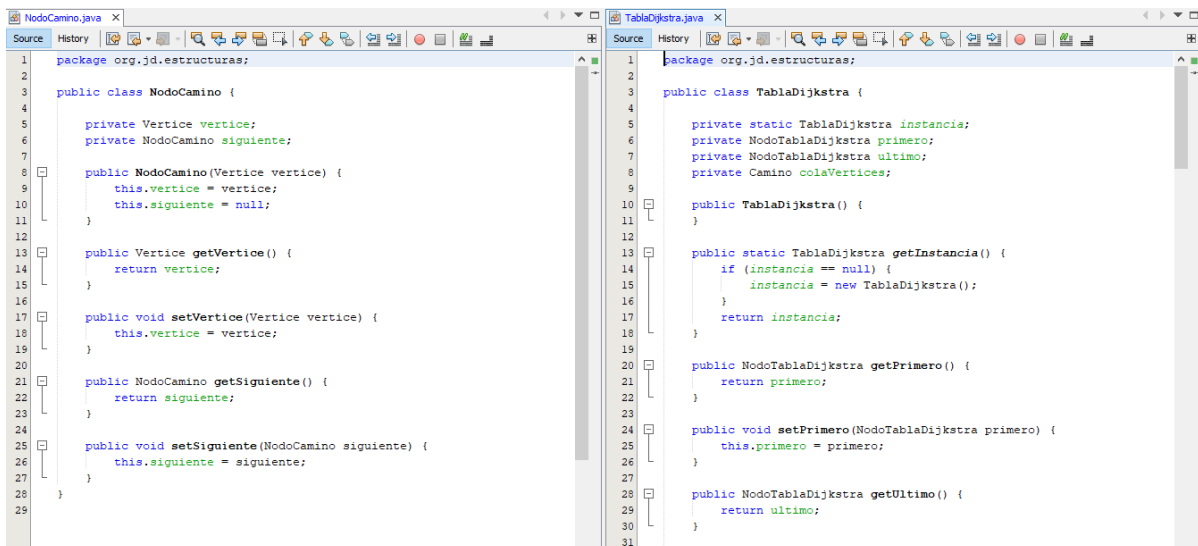
    private static ListaDoble instancia;
    private NodoListaDoble primero;
    private NodoListaDoble ultimo;
    private Encriptamiento encriptar;

    public ListaDoble() {
        primero = null;
        ultimo = null;
        encriptar = new Encriptamiento();
    }

    public static ListaDoble getInstancia() {
        if (instancia == null) {
            instancia = new ListaDoble();
        }
        return instancia;
    }

    public NodoListaDoble getPrimero() {
        return primero;
    }

    public void setPrimero(NodoListaDoble primero) {
        this.primero = primero;
    }
}
```



The screenshot shows two IDE windows side-by-side. The left window displays the code for the `NodoCamino` class, and the right window displays the code for the `TablaDijkstra` class.

```
package org.jd.estructuras;

public class NodoCamino {

    private Vertice vertice;
    private NodoCamino siguiente;

    public NodoCamino(Vertice vertice) {
        this.vertice = vertice;
        this.siguiente = null;
    }

    public Vertice getVertice() {
        return vertice;
    }

    public void setVertice(Vertice vertice) {
        this.vertice = vertice;
    }

    public NodoCamino getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoCamino siguiente) {
        this.siguiente = siguiente;
    }
}
```

```
package org.jd.estructuras;

public class TablaDijkstra {

    private static TablaDijkstra instancia;
    private NodoTablaDijkstra primero;
    private NodoTablaDijkstra ultimo;
    private Camino colaVertices;

    public TablaDijkstra() {
    }

    public static TablaDijkstra getInstancia() {
        if (instancia == null) {
            instancia = new TablaDijkstra();
        }
        return instancia;
    }

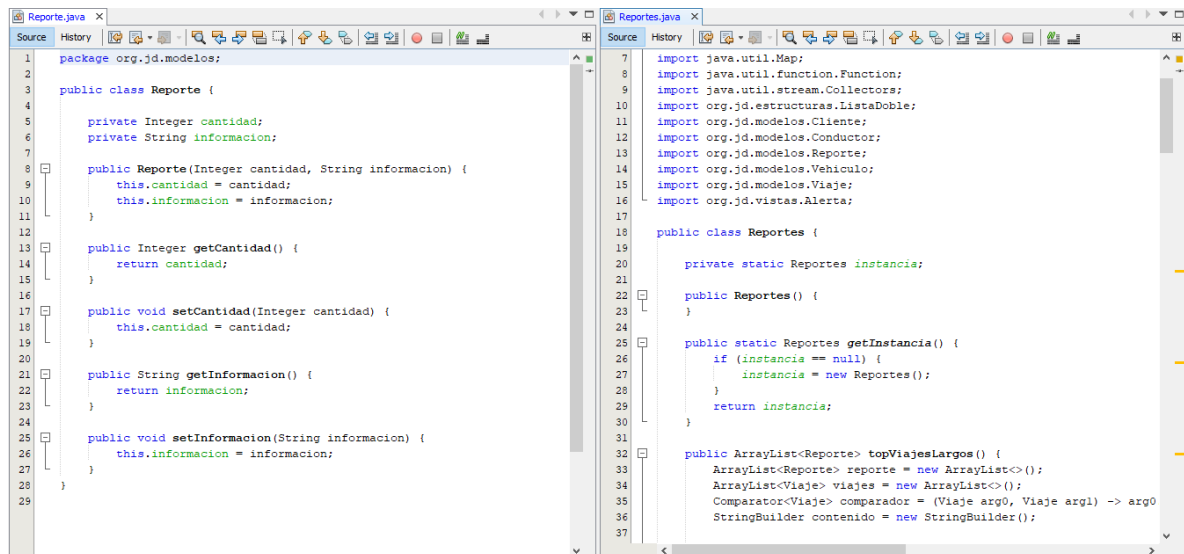
    public NodoTablaDijkstra getPrimero() {
        return primero;
    }

    public void setPrimero(NodoTablaDijkstra primero) {
        this.primero = primero;
    }

    public NodoTablaDijkstra getUltimo() {
        return ultimo;
    }
}
```

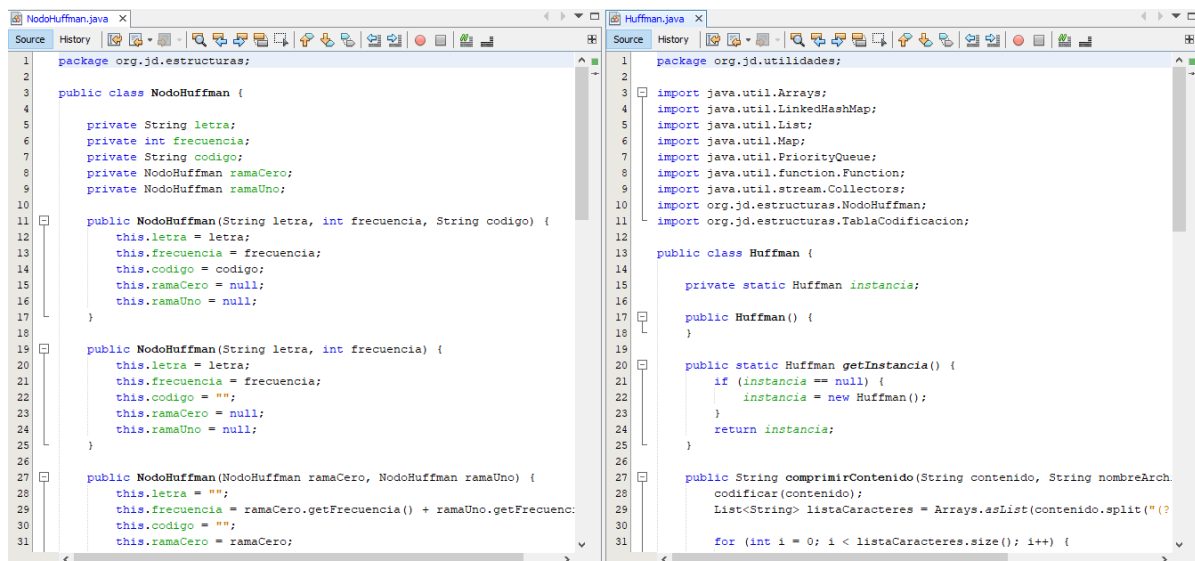

REPORTES Y TOPS

Para el manejo de cada uno de los reportes se utilizaron listas dinámicas las cuales se ordenaron por medio de una interfaz comparadora. Para cada uno de los reportes se crea un archivo de extensión edd, el cual por medio del método de Huffman se comprime y genera un archivo de salida de la tabla de codificación la cual contiene el carácter, frecuencia y código del mismo.



```
Reporte.java
1 package org.jd.modelos;
2
3 public class Reporte {
4     private Integer cantidad;
5     private String informacion;
6
7     public Reporte(Integer cantidad, String informacion) {
8         this.cantidad = cantidad;
9         this.informacion = informacion;
10    }
11
12    public Integer getCantidad() {
13        return cantidad;
14    }
15
16    public void setCantidad(Integer cantidad) {
17        this.cantidad = cantidad;
18    }
19
20    public String getInformacion() {
21        return informacion;
22    }
23
24    public void setInformacion(String informacion) {
25        this.informacion = informacion;
26    }
27
28 }
29

Reportes.java
7 import java.util.Map;
8 import java.util.function.Function;
9 import java.util.stream.Collectors;
10 import org.jd.estructuras.ListaDoble;
11 import org.jd.modelos.Cliente;
12 import org.jd.modelos.Conductor;
13 import org.jd.modelos.Reporte;
14 import org.jd.modelos.Vehiculo;
15 import org.jd.modelos.Viaje;
16 import org.jd.vistas.Alerta;
17
18 public class Reportes {
19     private static Reportes instancia;
20
21     public Reportes() {
22     }
23
24     public static Reportes getInstancia() {
25         if (instancia == null) {
26             instancia = new Reportes();
27         }
28         return instancia;
29     }
30
31     public ArrayList<Reporte> topViajesLargos() {
32         ArrayList<Reporte> reporte = new ArrayList<>();
33         ArrayList<Viaje> viajes = new ArrayList<>();
34         Comparator<Viaje> comparador = (Viaje arg0, Viaje arg1) -> arg0
35         StringBuilder contenido = new StringBuilder();
36
37     }
```



```
NodoHuffman.java
1 package org.jd.estructuras;
2
3 public class NodoHuffman {
4     private String letra;
5     private int frecuencia;
6     private String codigo;
7     private NodoHuffman ramaCero;
8     private NodoHuffman ramaUno;
9
10    public NodoHuffman(String letra, int frecuencia, String codigo) {
11        this.letra = letra;
12        this.frecuencia = frecuencia;
13        this.codigo = codigo;
14        this.ramaCero = null;
15        this.ramaUno = null;
16    }
17
18    public NodoHuffman(String letra, int frecuencia) {
19        this.letra = letra;
20        this.frecuencia = frecuencia;
21        this.codigo = "";
22        this.ramaCero = null;
23        this.ramaUno = null;
24    }
25
26    public NodoHuffman(NodoHuffman ramaCero, NodoHuffman ramaUno) {
27        this.letra = "";
28        this.frecuencia = ramaCero.getFrecuencia() + ramaUno.getFrecuencia();
29        this.codigo = "";
30        this.ramaCero = ramaCero;
31        this.ramaUno = ramaUno;
32    }
33
34 }
35

Huffman.java
1 package org.jd.utilidades;
2
3 import java.util.Arrays;
4 import java.util.LinkedHashMap;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.PriorityQueue;
8 import java.util.function.Function;
9 import java.util.stream.Collectors;
10 import org.jd.estructuras.NodoHuffman;
11 import org.jd.estructuras.TablaCodificacion;
12
13 public class Huffman {
14     private static Huffman instancia;
15
16     public Huffman() {
17     }
18
19     public static Huffman getInstancia() {
20         if (instancia == null) {
21             instancia = new Huffman();
22         }
23         return instancia;
24     }
25
26     public String comprimirContenido(String contenido, String nombreArch) {
27         codificar(contenido);
28         List<String> listaCaracteres = Arrays.asList(contenido.split("(?"));
29         for (int i = 0; i < listaCaracteres.size(); i++) {
30
31         }
```