Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Estructuras de Datos
Sección “A”

“MANUAL TÉCNICO”

Samuel Alejandro Pajoc Raymundo

Carné: 201800665

Objetivos

General:

Brindar al lector una guía que contenga la información del manejo de clases, atributos, métodos y del desarrollo de la aplicación de consola, de modo que sea posible las actualizaciones y modificaciones futras.

Específicos:

- Mostrar al lector una descripción lo más completa y detallada posible del SO, IDE entre otros utilizados para el desarrollo de la aplicación.

- Proporcionar al lector una explicación técnica - formal de los procesos y relaciones entre métodos y atributos que conforman la parte operativa de la aplicación.

Introducción

Este manual técnico posee como finalidad dar a conocer al lector que pueda requerir hacer modificaciones futuras al software el desarrollo de la aplicación “Llega Rapidito” indicando el IDE utilizado para su creación, las librerías implementadas, su versión y requerimientos del sistema.

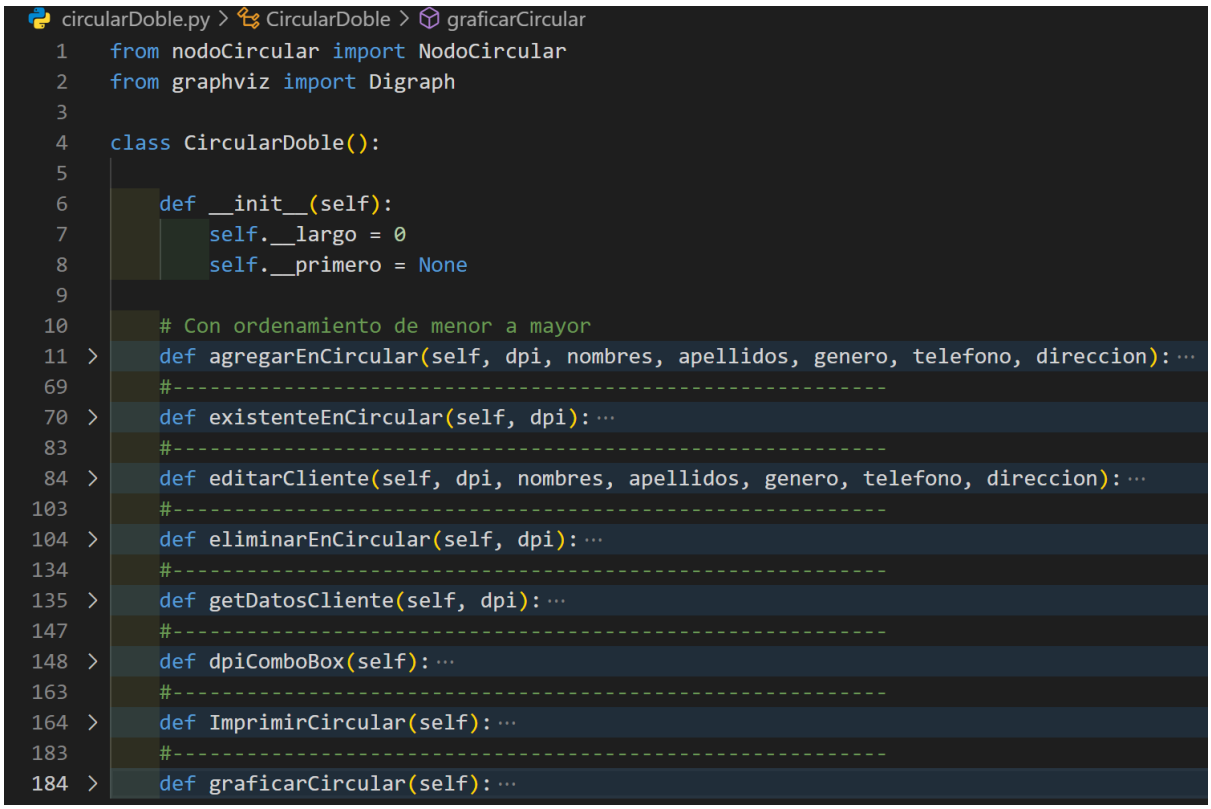
La aplicación tiene como objetivo leer los datos ingresados por los gerentes de empresas mediante un entorno gráfico, y observar por medio de distintos tipos de reportes la estructuración y el flujo de todos los datos. La aplicación permitirá tomar decisiones importantes que afectarán el rumbo de una empresa aportando resultados de procesos de entrega de artículos para obtener mayor rendimiento y productividad.

Descripción de la Solución

Para el almacenamiento de los Clientes, se utilizó una lista doblemente enlazada, la cual se entra estructurada de 1 clase llamada nodoCircular y una clase llamada CircularDoble; en la clase nodoCircular, se crea un objeto en el cual se almacenan todos los datos pertenecientes al Cliente, el primer nodo cliente creado, inicia siendo el nodo con puntero primero, y además se cuenta con una variable de tipo privada, la cual permite conocer el largo de la lista, dado que si está aún no cuenta con suficientes nodos, su eliminación y su inserción seguirán siendo distintas, a cuando ya tiene más de un dato. Cada nodo posee dos punteros, uno que lleva hacia la posición siguiente y otro que permite retroceder a la posición anterior, ambos se inicializan en None, lo cual en Python significa nulo. A continuación, se presenta una imagen del código perteneciente a la clase NodoCircular, junto a la Clase Circular, la cual le da vida a la lista doblemente enlazada:

```python
nodoCircular.py > NodoCircular
class NodoCircular():
    def __init__(self, dpi, nombres, apellidos, genero, telefono, direccion):
        self.__dpi = dpi
        self.__nombres = nombres
        self.__apellidos = apellidos
        self.__genero = genero
        self.__telefono = telefono
        self.__direccion = direccion
        self.__drchaCircular = None
        self.__izqCircular = None

    #Getters
    def getDPI(self):
        return self.__dpi

    def getNombres(self):
        return self.__nombres

    def getApellidos(self):
        return self.__apellidos

    def getGenero(self):
        return self.__genero

    def getTelefono(self):
        return self.__telefono

    def getDireccion(self):
        return self.__direccion

    def getDrchaCircular(self):
        return self.__drchaCircular

    def getIzqCircular(self):
        return self.__izqCircular

    #Setters
    def setDPI(self, dpi):
        self.__dpi = dpi

    def setNombres(self, nombres):
        self.__nombres = nombres

    def setApellidos(self, apellidos):
        self.__apellidos = apellidos

    def setGenero(self, genero):
        self.__genero = genero

    def setTelefono(self, telefono):
        self.__telefono = telefono

    def setDireccion(self, direccion):
        self.__direccion = direccion

    def setDrchaCircular(self, drchaCircular):
        self.__drchaCircular = drchaCircular

    def setIzqCircular(self, izqCircular):
        self.__izqCircular = izqCircular
```

Dentro de la clase CircularDoble se puede encontrar todos los métodos que cran la estructura de lista doblemente enlazada, por lo que es aquí donde se realiza la inserción (enlace) de los nodos con respecto al nodo primero, el cual también es un nodo de la clase nodoCircular, el cual será el que nos indique a partir de donde es que se inicia a recorrer la lista circular.



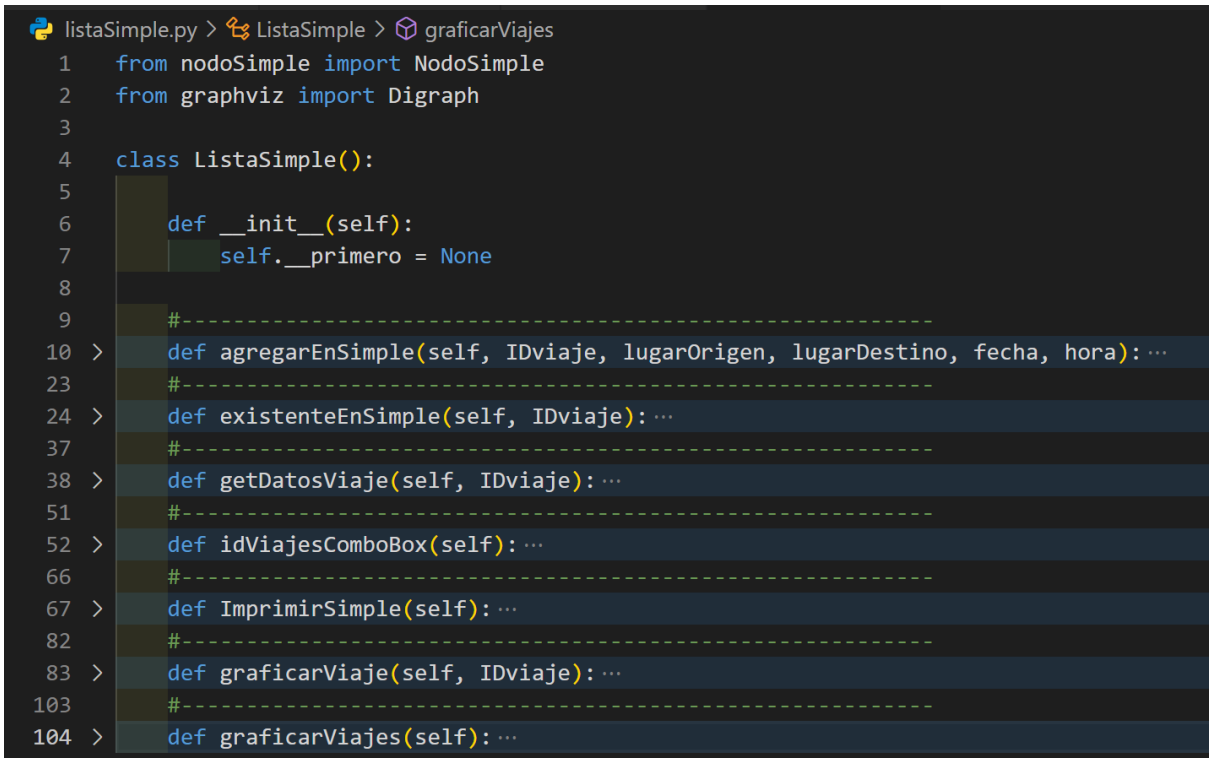
La estructura que se pensó para el almacenamiento de los vehículos fue un Árbol B, pero lamentablemente no se pudo proseguir con su implementación, por lo que el código avanzado queda a disposición de cualquier persona que quiera realizar dicha implementación.

Por otra parte, los viajes se almacenaron dentro de una lista simplemente enlazada, la cual, al igual que la lista doble, esta está hecha a base de otra clase llamada nodoSimple, el cual conserva todos los datos correspondientes a los viajes, y así, la clase lista Simple, únicamente se encarga de administrar la inserción de los nodos con la información, a continuación, se mostrará el diseño del nodoSimple:

nodoSimple.py > NodoSimple > setSig

```python
class NodoSimple():

    def __init__(self, IDviaje, lugarOrigen, lugarDestino, fecha, hora):
        self.__IDviaje = IDviaje
        self.__lugarOrigen = lugarOrigen
        self.__lugarDestino = lugarDestino
        self.__fecha = fecha
        self.__hora = hora
        self.__sig = None

    #Getters
    def getIDviaje(self):
        return self.__IDviaje

    def getLugarOrigen(self):
        return self.__lugarOrigen

    def getLugarDestino(self):
        return self.__lugarDestino

    def getFecha(self):
        return self.__fecha

    def getHora(self):
        return self.__hora

    def getSig(self):
        return self.__sig

    #Setters
    def setIDviaje(self, IDviaje):
        self.__IDviaje = IDviaje

    def setLugarOrigen(self, lugarOrigen):
        self.__lugarOrigen = lugarOrigen

    def setLugarDestino(self, lugarDestino):
        self.__lugarDestino = lugarDestino

    def setFecha(self, fecha):
        self.__fecha = fecha

    def setHora(self, hora):
        self.__hora = hora

    def setSig(self, siguiente):
        self.__sig = siguiente
```

Todos los métodos que dan vida a la estructura de la lista simplemente enlazada se muestran a continuación, pero para mayor consulta, siempre se puede consultar el código fuente directamente:



Esta clase se encuentra diseñada primordialmente por un puntero que apunta hacia el primer dato a ser insertado, de modo que tengamos una referencia de donde es que se debe iniciar con el recorrido, no es necesario tener un contador de tamaño como con la lista doble, dado que este al poseer un final apuntando a None, siempre se podrá encontrar el fin y por ende se tiene una referencia de hasta donde debe llegar cualquier recorrido para poder realizar su correspondiente validación.

Los métodos para graficar las estructuras, son las siguientes:

Para la lista Simple:

```
def graficarViajes(self):
    if(self.__primero != None):
        temp = self.__primero

        #creacion de grafo dirigido
        dot = Digraph(name='ListaViajes', graph_attr={'rankdir': 'LR'})
        while(temp != None):
            #Creacion nodo
            dot.node(temp.getIDviaje(), label='< <TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0"><TR><TD>'+temp.getIDviaje()+'</TD></TR> <TR><TD>'+temp.getLugarOrigen()

            #Creacion enlace
            if(temp.getSig() != None):
                dot.edge(temp.getIDviaje(), temp.getSig().getIDviaje())

            temp = temp.getSig()

        #Guarda y renderiza el grafico
        dot.render('Graph_Viajes', format='png', view=True)

    else:
        print("Lista Simple Vacia, no Graficar Estructura!")
```

Para la lista Doble:

```
184    def graficarCircular(self):
185        if(self.__primero != None):
186            temp = self.__primero
187            siguiente = None
188
189            #creacion de grafo dirigido
190            dot = Digraph(name='CircularClientes', graph_attr={'rankdir': 'LR'})
191
192            #Agrega nodos
193            while True:
194                dot.node(str(temp.getDPI()), label='< <TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0"><TR><TD>'+str(temp.getDPI())+'</TD></TR> <TR><TD>'+temp.getNombres()+'
195
196                #Agrega Enlaces dobles entre nodos
197                siguiente = temp.getDrchaCircular()
198                dot.edge(str(temp.getDPI()), str(siguiente.getDPI()))
199                dot.edge(str(siguiente.getDPI()), str(temp.getDPI()))
200
201                temp = temp.getDrchaCircular()
202                if(temp == self.__primero):
203                    break
204            #Guarda y renderiza el grafico
205            dot.render('Graph_Circular', format='png', view=True)   #view=True, permite ver automaticamente la imagen generada
206
207        else:
208            print("Lista Circular Vacia, no Graficar!")
```

Para graficar el mapa, el cual se carga en la ventana principal:

```
66    def graphMapa(rutas):
67        dot = Graph(engine='fdp')
68        nodosExistentes=[]
69
70        #Creacion nodos
71        for ruta in rutas:
72            if(ruta[0] not in nodosExistentes):
73                # .node(nombreNodo, etiquetaNodo, ...)
74                dot.node(ruta[0], ruta[0], shape='circle', style='filled', color='lightgreen')
75                nodosExistentes.append(ruta[0])
76
77            if(ruta[1] not in nodosExistentes):
78                dot.node(ruta[1], ruta[1], shape='circle', style='filled', color='lightgreen')
79                nodosExistentes.append(ruta[1])
80
81        # Agregar aristas (conexiones) entre nodos con etiquetas (distancias)
82        for ruta in rutas:
83            dot.edge(ruta[0], ruta[1], label=ruta[2])
84
85        dot.render('grafoRutasMapa', format='png')
86        actualizarMapa()
```

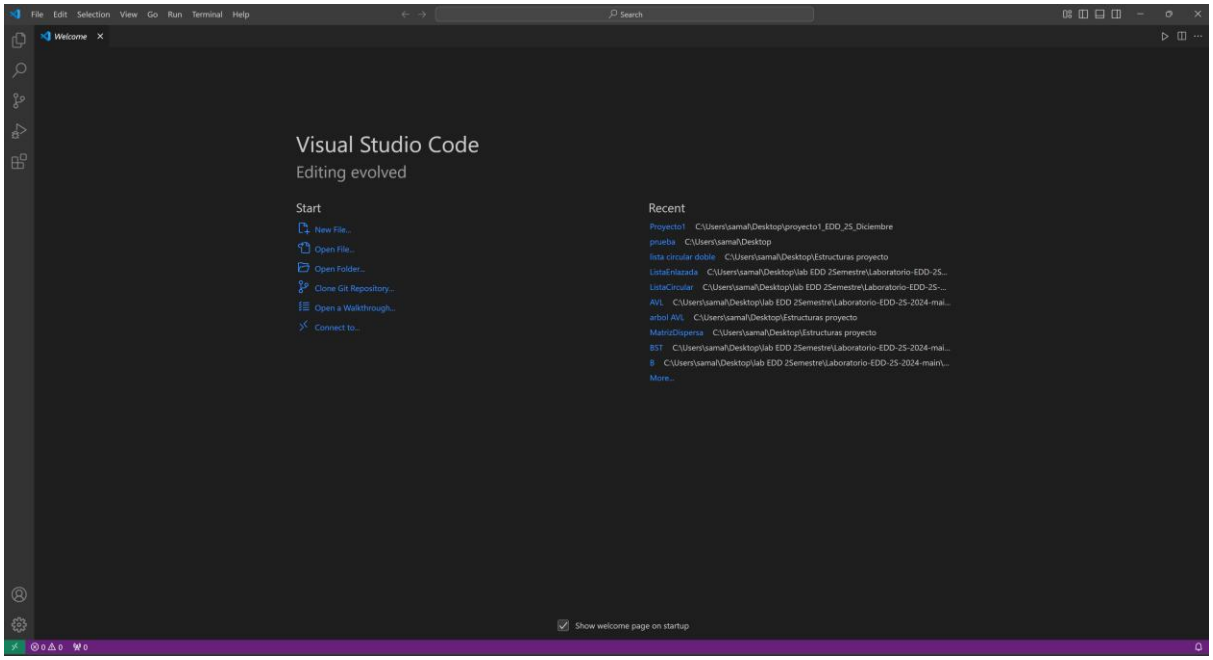
Por último, la ventana principal funciona en armonía con todos los componentes, gracias a la siguiente implementación:

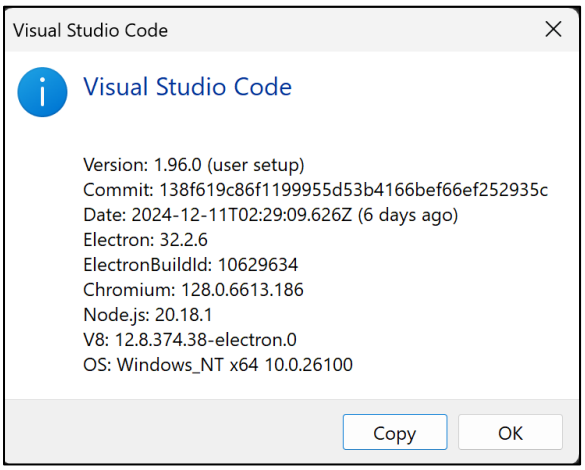
```
660  #=================================================================================
661  #========================== Ajustes ventana main ================================
662  mainWindow = Tk()
663  mainWindow.title("LLEGA RAPIDITO")
664  mainWindow.geometry("500x600")
665  mainWindow.config(bg='orange')
666  centrarVentana(mainWindow)
667  mainWindow.resizable(False, False)#Hace que no sea Redimensionable
668  #=================================================================================
669  #====================== CUERPO - MAIN ============================================
670  btnCargaMapa = Button(mainWindow, text="INICIAR", bg="red", command=cargaArchivoMapa) #PARA FUNCIONAMIENTO NORMAL, REMOVER ESTA LINEA
671  #btnCargaMapa = Button(mainWindow, text="INICIAR", bg="red", command=AbrirVentanaClientes) #PARA PROBAR VENTANA CLIENTES
672  #btnCargaMapa = Button(mainWindow, text="INICIAR", bg="red", command=AbrirVentanaVehiculos)  #PARA PROBAR VENTANA VEHICULOS
673  #btnCargaMapa = Button(mainWindow, text="INICIAR", bg="red", command=AbrirVentanaViajes)  #PARA PROBAR VENTANA VIAJES
674  btnCargaMapa.place(x=223, y=40)
675
676  btnClientes = Button(mainWindow, text="Clientes", bg="darkgreen", fg="white", command=AbrirVentanaClientes)
677
678  btnVehiculos = Button(mainWindow, text="Vehiculos", bg="darkgreen", fg="white", command=AbrirVentanaVehiculos)
679
680  btnViajes = Button(mainWindow, text="Viajes", bg="darkgreen", fg="white", command=AbrirVentanaViajes)
681
682  btnVerMapaClaro = Button(mainWindow, text="VER MAPA", bg="black", fg="white", command=verMapa)
683
684  btnReportes = Button(mainWindow, text="Reportes", bg="black", fg="white")
685
686  imagenLogo = Image.open("llegaRa.png")
687  nuevoSizeLogo = imagenLogo.resize((470,430), Image.LANCZOS)
688  paraMapa = ImageTk.PhotoImage(nuevoSizeLogo)
689
690  lbMapa = Label(mainWindow, image=paraMapa)
691  lbMapa.place(x=10, y=150)
692  #-----------------------------------------------------------------------
693  mainWindow.mainloop()
```

Todas las ventanas generadas, siguen el mismo patrón de implementación.

IDE

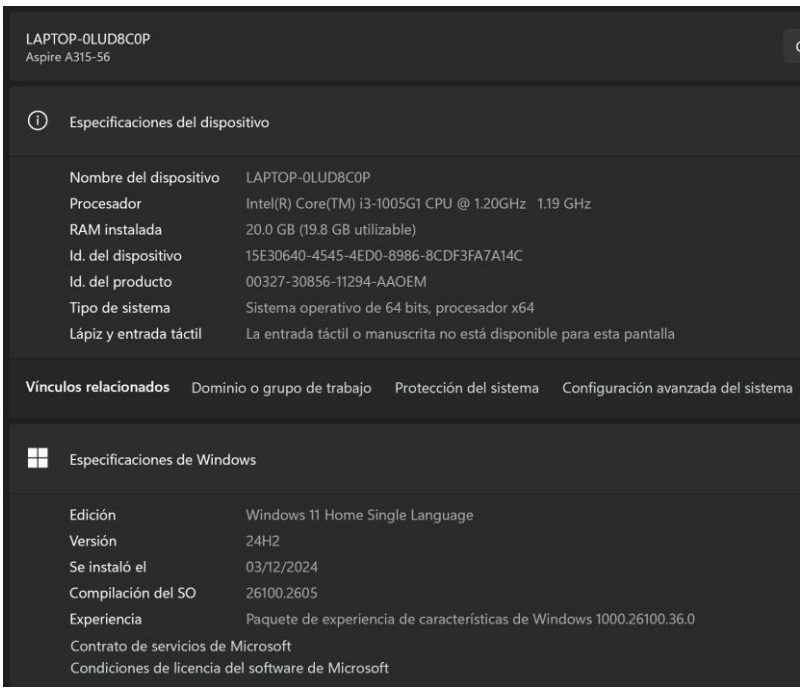
El IDE con el que se desarrolló el proyecto fue “Visual Studio Code” 1.96.0, dado que el desarrollo resultó más sencillo gracias a su apartado de extensiones, las cuales permiten tener un apoyo mayor al momento de generar el código fuente, dado que su soporte al desarrollador es muy bueno y agiliza considerablemente la producción.





Requerimientos:

- Sistema Operativo:
  El sistema operativo en el que se llevó a cabo la realización del proyecto fue Windows 11 de 64 bits.



Librerías Utilizadas

Las librerías utilizadas para el desarrollo de este proyecto fueron:

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
from tkinter import messagebox
from io import open
from PIL import Image, ImageTk
from graphviz import Graph
from datetime import datetime
import os
```