Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2022.

# PROYECTO 2: Programa de Simulación de Empresas y su Manejo de Clientes en sus Puntos de Atención.

**202109732 – José Eduardo Galdámez González**

## Resumen

Una parte bastante importante para entender las posibilidades dentro la programación es lo diverso que se puede trabajar, pero hay conceptos bastante importantes en nuestro ambiente, tomando en cuenta que parte del programador es renovarse continuamente, en el siguiente ensayo resumiremos lo mejor posible un reto diferente.

En este caso volveremos en los TDA, que son como dicen sus siglas un tipo de dato abstracto, se le conoce también como un modelo matemático por su contenido que es un conjunto de operaciones a realizar para un grupo de datos. Existen cuatro tipos de TDA que son: TDA lista, TDA pila, TDA cola y TDA cola de prioridad. En este caso aplicamos principalmente la lista y la pila, junto con esto un buen manejo de archivo XML y su obtención de la data. También se debe de resaltar la implementación de la librería para grafica de Graphviz.

## Palabras clave

TDA pila.

Graphviz.

TDA lista.

## *Abstract*

*A very important part to understand the possibilities in programming is the diversity that can be worked, but there are very important concepts in our environment, taking into account that part of the programmer is to renew continuously, in the following essay we will summarize as best as possible a different challenge.*

*In this case we will return to the TDA, which are as they say their acronym an abstract data type, it is also known as a mathematical model for its content which is a set of operations to be performed for a set of data. There are four types of TDA which are: TDA list, TDA stack, TDA queue and TDA priority queue. In this case we apply mainly the list and the stack, together with a good XML file handling and data retrieval. The implementation of the Graphviz graph library should also be highlighted.*

## *Keywords*

TDA list.

Graphviz.

TDA stack.

## Introducción

Parte importante de la programación es aplicar la lógica y el uso de todo nuestro conocimiento para el trabajo de nuestro proyecto principalmente cuando hablamos de lo que es el uso de los TDA, ya que, si se estructura bien, a la hora de su uso, será más fácil implementarlo, con esto también el hecho de estructurar un buen manejo de archivos XML nos ayuda a que, a la hora de necesitar información necesaria, no habrá problema.

Con estos conceptos podemos hablar un poco el cómo se quiere lograr el proyecto, ya que este es un simulador el cual obtendrá información desde 2 configuración tipo XML, y con eso junto a nuestro diagrama de clases ir estructurando la información para que después poder ejecutar diversas opciones de manejo de Empresa y Puntos de Atención, finalmente llegando a la parte más importante que es la simulación la cual tendrá lugar de manera de código o grafica la cual se trabajó con Graphviz.

## Desarrollo del tema

Para generalizar hablaremos acerca de cómo dividiremos esta sección, para comenzar el objetivo del proyecto ya explicado en la introducción iremos desglosando por partes, primero en cómo se trabajó nuestro diagrama de clases, junto a eso el cómo lo dividimos en código, también se adentrara a lo que es el manejo de archivos XML con la forma correcta de trabajo para la obtención de la data y como estructurar de la manera más efectiva y finalmente explicaremos a detalle el TDA que se utilizó para cada parte de nuestro programa y el cómo fuimos dándole forma, teniendo en cuenta que se utilizó la lista simple y la pila.

En lo que fue nuestra base del proyecto siempre es muy importante el cómo se fue trabajando dentro del Código, dando las bases, lo cual es primero detectar y analizar el objetivo del programa y estructurarlos según sea dado, en este al entender que ser el manejo de empresas y de sus puntos de atención identificamos que era necesario hacer un diagrama el cual estaría basado en darle una forma y estructura a la hora de ingresarle información y manejarla.

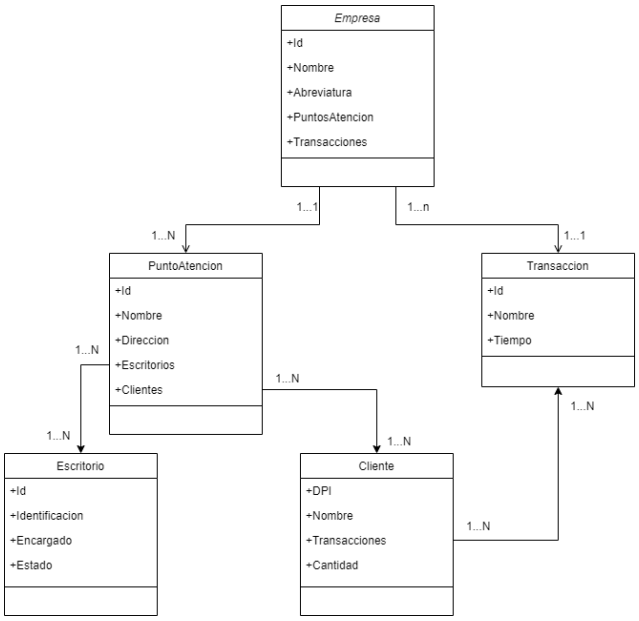


*Figura 1.* Imagen del Diagrama de Clases

Fuente: Propia

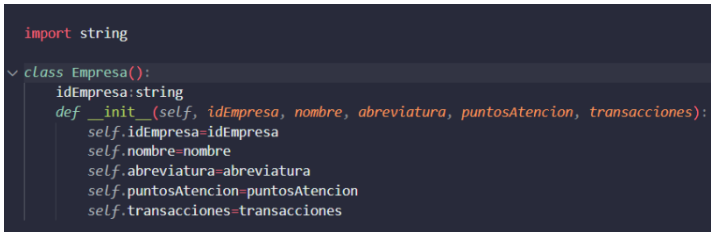


*Figura 2.* Imagen la clase Empresa basado en el diagrama

Fuente: Propia

Ya que teníamos nuestra bases para darle información y que este pueda trabajarla, comenzamos con el manejo de archivo con las respectivas librerías para trabajarlo, con esto estructuramos nuestro código para que este vaya desglosando cada parte del documento, seccionándolo según sea dado el formato de XML, ya desglosándolo según la lógica que estemos utilizando iremos dándole los valores de cada clase, en este caso se fue de atrás hacia adelante empezando el diagrama de forma contraria, para así ir haciendo las validaciones mas correctas y precisa la obtener la información de del XML.
Se debe de resaltar que se trabajó con la librería de xml.Etree, la cual incluye herramientas para analizar XML usando APIs basadas en eventos y documentos, buscando documentos analizados con expresiones XPath, y creando nuevos o modificando documentos existentes.


*Figura 3.* Imagen de los importes de las librerías.

Fuente: Propia


*Figura 4.* Imagen de Obtención de archivo.

Fuente: Propia


*Figura 5.* Imagen de cómo se fue trabajando la manipulación del archivo xml.

Fuente: Propia

La lista es una estructura de datos muy importante en los lenguajes de programación donde representa una colección de elementos ordenados, puede contener elementos repetidos y cada elemento de la lista tiene un índice que lo ubica dentro de la misma, con esto nos ayudo a trabajar el proceso de guardado de información.
Recordando que la implementación muy común de lista es la llamada lista enlazada, en donde se representa internamente como Nodos que referencian al siguiente. El proyecto ya con la previa estructuración del diagrama de clases fuimos utilizando listas simples para el uso de información, en este caso solo creamos una clase Nodo junto a sus funciones de la lista simple. Ya con esto se trabajo el guardado de información con agregar al final ya que se estaba haciendo de atrás hacia delante este nos iba a beneficiar, ya que no alteraría el orden de la información del archivo.

En la lista enlazada, agregar un elemento es fácil, o poco costoso en términos de locación de memoria. Simplemente debemos alocar espacio para un nuevo nodo, y actualizar el ultimo nodo, para que apunte al nuevo, como "siguiente".

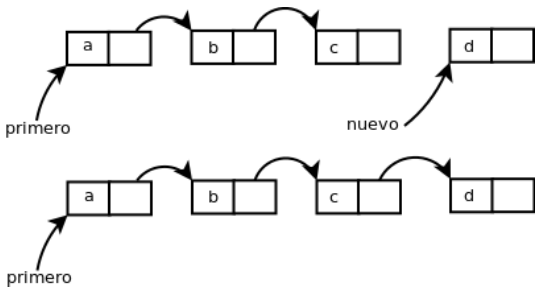


*Figura 6.* Imagen del funcionamiento de agregar al final en la lista enlazada.

Fuente: Google Sites

```
class Nodo():
    def __init__(self, dato= None, siguiente = None):
        self.dato = dato
        self.siguiente = siguiente
```

*Figura 7.* Imagen de la clase Nodo.

Fuente: Propia

```
from Nodo import *
class Lista_simple():
    def __init__(self):
        self.cabeza = None
        self.ultimo = None
        self.size = 0

    def vacio(self):
        return self.cabeza == None

    def agregar_al_final(self, dato):
        if self.vacio():
            self.cabeza = self.ultimo = Nodo(dato)
        else:
            aux = self.ultimo
            self.ultimo = aux.siguiente = Nodo(dato)
            self.ultimo.anterior = aux
        self.size +=1

    def size(self):
        return self.size
    def vaciar(self):
        self.cabeza=None
```

*Figura 8.* Imagen de la clase de la Lista Simple.

Fuente: Propia

Por último, tenemos el método FIFO (First In - First Out): Primero en entrar - Primero en salir, el cual nos ayudo en poder simular lo que era la cola de clientes, este mismo se le denomina como una pila (stack) que es un objeto similar a una pila de platos, donde se puede agregar y sacar datos sólo por el extremo superior. En computación esta estructura es utilizada ampliamente, aunque muchas veces los usuarios ni siquiera se percaten. Con este entendimiento modelamos nuestro algoritmo para que este actúa como una cola en constante actualización.

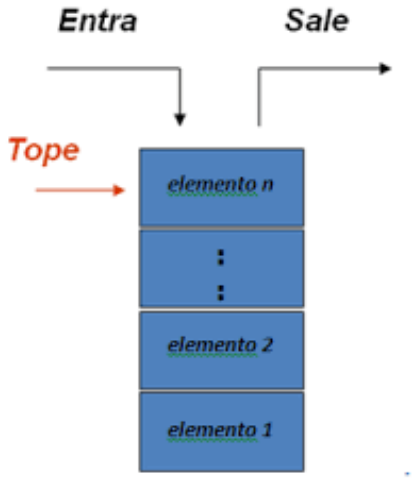


*Figura 9.* Funcionamiento de la pila (FIFO).

Fuente: Google Sites

## Conclusiones

El proyecto en general fue bastante complejo con tanta abstracción al diseñar nuestras funciones, pero apartado técnico es una de las cosas más desafiantes e interesantes, cuando se habla en el manejo de archivos XML, se tuvo que diseñar una forma óptima y útil para estructurar correctamente el código y la división de información según nuestro diagrama de clases, me parece muy importante que pongamos en practica todo estos conceptos y temas, ya que a futuro el manejo de estos casos será más fácil.

También resaltar lo capaz y diverso que es el mundo del TDA, tiene tantas formas de trabajar y aplicar

nuestra propia lógica que se vuelve un constante uso de lógica y programación que ayuda al estudiante a desarrollar las habilidades de resolución de cualquier tipo de problema.
En general el Proyecto No. 2, ha sido un reto positivo a lo mucho que se puede hacer en base a un problema planteado y el uso de herramientas como las listas enlazadas, y el saber graficarlas en base a librerías hacen que sea mas un reto personal que un simple programa.

## Referencias bibliográficas

Allen Downey, Jeffrey Elkner, y Chris Meyers. (2002) Aprenda a Pensar Como un Programador. Wellesley, Massachusetts

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford Introductions to Algorithms (2003). MIT Press. ISBN 0-262-03293-7, pp. 205–213, 501–505

Donald Knuth. Fundamental Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Sections 2.2.3–2.2.5, pp.254–298.

Alan Terraza. (6 de marzo, 2021). Algoritmo FIFO. Fuente:https://es.scribd.com/document/500445705/Algoritmo-fifo

Antonakos, James L. and Mansfield, Kenneth C., Jr. Practical Data Structures Using C/C++ (1999). Prentice Hall. ISBN 0-13-280843-9, pp. 165–190