Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 1er. Semestre 2022.

Gestion de empresas y atención de clientes

Carnet 201445840 – Luis Humberto Lémus Pérez

Resumen

La presente documentación describe las distintas funcionalidades, métodos, y validaciones que se utilizaron para la realización de la aplicación, la aplicación tienen como objetivo el análisis y la configuración que se obtienen de un archivo .xml; en el cual se lee obteniendo no solo los datos de las empresas sino también la información de cada cliente, escritorios activos y transacciones por cliente

Palabras clave

Paradigma
Nodo
Lista simple enlazada
Metodos
Clases
Interfaz de usuario
Polimorfismo
Modularidad
Herencia

Abstract

This documentation describes the different functionalities, methods, and validations that were used to make the application, the application aims at the analysis and configuration that are obtained from a file .xml; in which it is read obtaining not only the data of the companies but also the information of each client, active desktops and transactions per client

Keywords

Paradigm

Node

Single Linked List

Methods

Classes

User Interface

Polymorphism

Modularity

Inheritance

Introducción

Describe las funciones y que paradigma de programación se utilizó así como la lógica utilizada para la solución del problema también se indica los diagramas de clases y se explica la utilización de listas enlazadas para guardar la información requerida también generar múltiples procesos.

Se Describe las validaciones utilizadas para mostrar los errores que pudiesen ocurrir así como la utilización de la aplicación.

Desarrollo del tema

a. Explicación de nodos y listas simples

Nodo: un nodo contiene un dato y un apuntador al siguiente nodo

*Figura 1.*Nodo de una lista simple.

Lista simple:

Una lista es una secuencia de elementos del mismo tipo o clase almacenados en memoria. Las listas son estructuras lineales, donde cada elemento de la lista, excepto el primero, tiene un único predecesor y cada elemento de la lista, excepto el último, tiene un único sucesor

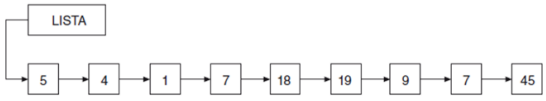


*Figura 2.*representacion de una lista simple.

Metodo POP

En una lista simple es capaz de eliminar el primer nodo asu vez de retornar el dato para su posterior utilizacion.

```
def pop(self):
    if self.cabeza is None:
      return None

    temp = self.cabeza
    self.cabeza = self.cabeza.siguiente
    return temp.dato
```

También es capaz de eliminar su último nodo que a su vez retornara el dato para su posterior implementación

```
def popultimo(self):
    temp:Nodo
    if self.cabeza is None:
      return None
    else:
      temp=self.cabeza
      while temp.siguiente.siguiente is not
None:
        temp=temp.siguiente
      escri=temp.siguiente.dato
      temp.siguiente=None
      return escri
```

b. Paradigma de programación utilizado

Programación orientada a objetos:

- Polimorfismo
- Modularidad
- Herencia

Paradigma Imperativo

- Programación modular

c. Diagrama de clases

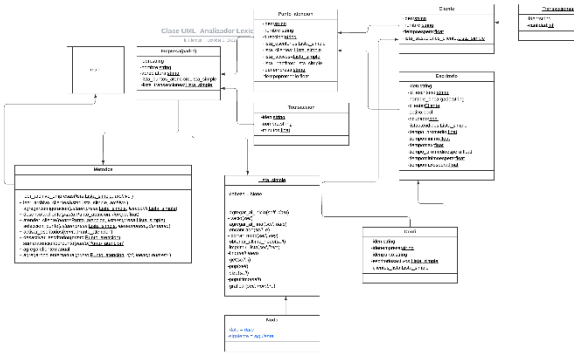


Figura 3.Diagrama de clases.

d. Métodos Principales utilizados

```
def agregarconfiguracion(listaempresa:Lista_simple, listaconfi:Lista_simple):
    tamanio=listaconfi.tamanio
    #print(tamanio)
    confi:Confi
    i=0
    for i in range(tamanio+1):
        print(i)
        confi=listaconfi.get(i)
        #print(confi.idenempresa)
        encontrado=listaempresa.find(confi.idenempresa)
        if encontrado!=None:

            encontrado1=encontrado.dato.lista_puntos_atencion.find(confi.idenpunto)
            if encontrado1!=None:
                #print("algo")
                #print(confi.clientes_lista.tamanio)
                for i in range(confi.clientes_lista.tamanio+1):
                    cleinte=confi.clientes_lista.pop()
                    if cleinte!=None:
                        encontrado1.dato.lista_clientes.agregar_al_final(cleinte)
                for i in range(confi.escritoriosactivos.tamanio+1):
                    escri=confi.escritoriosactivos.get(i)
                    encontrado2=encontrado1.dato.lista_escritorios.find(escri)
                    if encontrado2!=None:
                        encontrado2.dato.activo=True
                        encontrado1.dato.lista_activos.agregar_al_final(encontrado2
```

Figura 5. Método agrega la configuración del archivo configuración inicial

```
def seleccion_punto(listaempresa:Lista_simple, idenempresa,idenpunto):
    encontrado=listaempresa.find(idenempresa)
    iden=encontrado.dato.iden
    if encontrado!=None:
        encontrado1=encontrado.dato.lista_puntos_atencion.find(idenpunto)
        encontrado1.dato.idenempresa=iden
        return encontrado1.dato
```

Figura 6. Método selecciona un punto de atención para su posterior utilización

```
def desencolarcliente(punto:Punto_atencion, tiempo:float):
    escritorio:Escritorio
    cliente:Cliente
    for i in range(punto.lista_activos.size()):
        escritorio=punto.lista_activos.get(i)
        if escritorio.activo==True and escritorio.ocupado==False:
            cliente=punto.lista_clientes.pop()

            if cliente!=None:
                escritorio.ocupado=True
                print(cliente.iden, cliente.nombre)
                cliente.tiempoespera=tiempo
                escritorio.cliente=cliente
                #cliente.lista_trasaciones_cliente.imprimir_lista("iden")
                #print(escritorio.cliente.iden)
            else:
                messagebox.showinfo("advertencia","ya no hay clientes que tender")
```

Figura 7. Método desencola un cliente a escritorios que estén activos

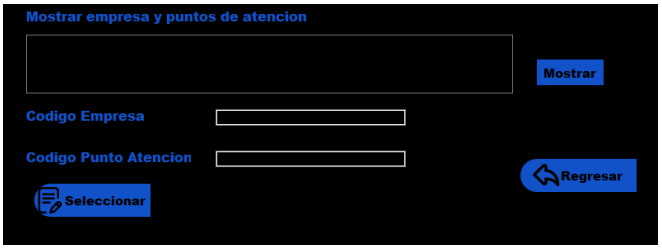
```
def atender_cliente(punto:Punto_atencion, listaempresa:Lista_simple):
    escritorio:Escritorio
    operacion:Transacciones_cliente
    tiempoop=0
    for i in range(punto.lista_activos.size()):
        escritorio=punto.lista_activos.get(i)
        if escritorio.activo==True and escritorio.ocupado==True and escritorio!=None:

            for i in range(escritorio.cliente.lista_trasaciones_cliente.size()):
                operacion=escritorio.cliente.lista_trasaciones_cliente.get(i)

                encontrado=listaempresa.find(punto.idenempresa)

                if encontrado!=None:
                    encontrado1= encontrado.dato.lista_transacciones.find(operacion.iden)
                    if encontrado1!=None:
                        minutos=encontrado1.dato.minutos
                        op=operacion.cantidad
                        tiempoop=float(minutos)*op+tiempoop
                        escritorio.ocupado=False
            iden=escritorio.cliente.iden
            nombre=escritorio.cliente.nombre
            tiempo=escritorio.cliente.tiempoespera
            escritorio.listaatendidos.agregar_al_final(Atendido(iden,nombre,tiempoop,tiempo))
            escritorio.cliente=Cliente("","")
            break
    return tiempoop
```

Figura 7. Método atiende un cliente y obtiene su tiempo de atención

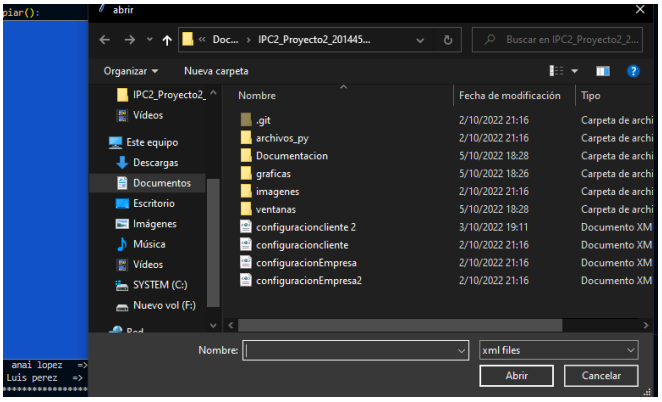
e. Interfaces de usuario

Figura 8.interfaz principal

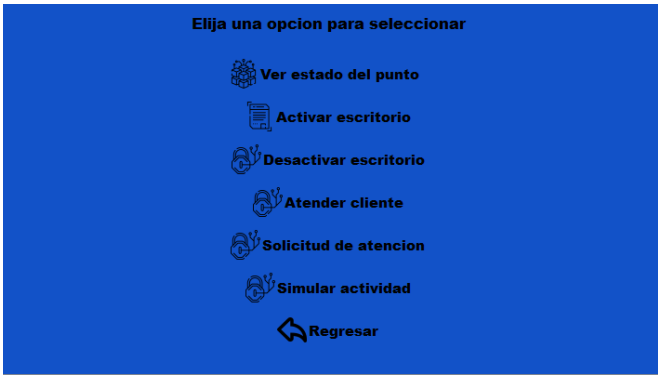
Figurara 8.seleccion de configuración empresa



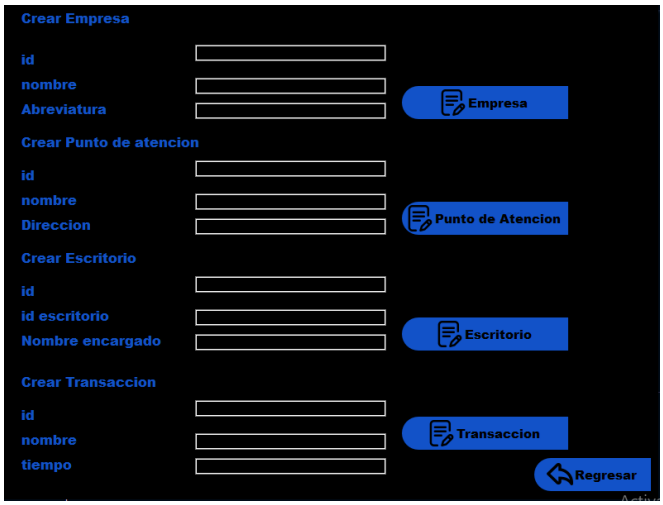
Figurara 11.Seleccion de punto de atencion



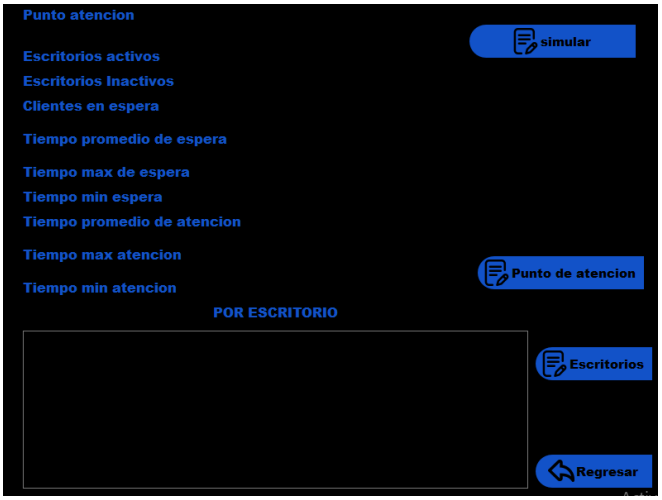
Figurara 9.seleccion archivo sml de confisistema y confinical



Figurara 12.menu manejo de puntos



Figurara 10.crea una empresa de forma manual



Figurara 12.simula la atencion de clientes

Conclusiones

El programa es capaz de a través de dos archivos de entrada xml guardar los datos de la empresa y configuración inicial en listas simples y dentro de ellas también guardar puntos de atención así como sus escritorios y los clientes que pertenecen al punto de atención.

También es capaz de seleccionar un punto de atención para hacer las distintas gestiones, a su vez de mostrar el estado de cada punto de atención, atender clientes, activar escritorios, desactivar escritorios, solicitud de atención simular atención.