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 2025

PROYECTO 2

NextGen Systems

202111134 Mario René Mérida Taracena

Resumen

El programa simula un sistema de atención al cliente para empresas utilizando Programación Orientada a Objetos (POO) en Python. Emplea estructuras de datos personalizadas como ListaEnlazada, DiccionarioPersonalizado y L istaDoblementeEnlazada para gestionar empresas, puntos de atención, escritorios y transacciones. El sistema permite cargar configuraciones desde archivos XML, asignar clientes a colas FIFO, activar/desactivar escritorios y simular el paso del tiempo para avanzar en la atención. Calcula automáticamente tiempos de espera y genera reportes gráficos con estadísticas de desempeño. La interfaz gráfica (Tkinter) facilita la interacción, mientras que la modularidad y el manejo de excepciones destacan su robustez, demostrando cómo la POO y las estructuras personalizadas optimizan sistemas de gestión complejos.

Palabras clave

POO (Programación Orientada a Objetos): Paradigma que organiza el código en clases y objetos para modelar entidades del mundo real.
Listas Enlazadas: Estructuras de datos dinámicas donde cada elemento (nodo) contiene datos y un enlace al siguiente nodo.
Simulación de Colas: Modelado computacional de sistemas FIFO (First-In-First-Out) para gestionar procesos secuenciales, como atención al cliente.

Abstract

The program simulates a corporate customer service system using Object-Oriented Programming (OOP) in Python. It employs custom data structures such as LinkedList, CustomDictionary, and DoublyLinkedList to manage companies, service points, desks, and transactions. The system allows users to load configurations from XML files, assign customers to FIFO queues, activate/deactivate desks, and simulate the passage of time to advance service. It automatically calculates wait times and generates graphical reports with performance statistics. The graphical interface (Tkinter) facilitates interaction, while its modularity and exception handling enhance its robustness, demonstrating how OOP and custom structures optimize complex management systems.

Keywords

OOP (Object-Oriented Programming): A paradigm that organizes code into classes and objects to model real-world entities.
Linked Lists: Dynamic data structures where each element (node) contains data and a link to the next node.
Queuing Simulation: Computational modeling of FIFO (First-In-First-Out) systems to manage sequential processes, such as customer service.

Introducción

Este proyecto implementa un sistema de atención a clientes para empresas guatemaltecas, simulando la gestión de puntos de atención, escritorios de servicio y transacciones. El sistema permite:

- Gestionar múltiples empresas con sus respectivos puntos de atención
- Administrar escritorios de servicio y su estado (activo/inactivo)
- Procesar transacciones con diferentes tiempos de atención
- Asignar clientes a colas de espera y calcular tiempos estimados
- Generar reportes gráficos del estado del sistema
- Simular el paso del tiempo para avanzar en la atención

El sistema está desarrollado completamente en Python, utilizando programación orientada a objetos y estructuras de datos personalizadas, con una interfaz gráfica basada en Tkinter.

Desarrollo del Tema

Análisis de Clases

Estructuras de Datos Personalizadas

Nodo y ListaEnlazada

- Propósito: Implementación básica de una lista enlazada simple
- Atributos/Métodos clave:
    - agregar(dato): Añade un elemento al final de la lista
    - __iter__(): Permite iterar sobre los elementos
    - filtrar(condicion): Filtra elementos basados en una condición
    - pop(index): Elimina y retorna el elemento en el índice dado

NodoDoble y ListaDoblementeEnlazada

- Propósito: Implementación de lista doblemente enlazada para manejo FIFO
- Características especiales:
    - Permite iteración en ambos sentidos
    - Método eliminar_primer_cliente() para operaciones FIFO

DiccionarioPersonalizado

- Propósito: Implementación de diccionario usando hashing
- Características:
    - Usa ListaEnlazada para manejar colisiones
    - Implementa funciones hash personalizadas para diferentes tipos de datos
    - Soporta operaciones básicas de diccionario (agregar, obtener, __contains__

ConjuntoPersonalizado

- Propósito: Implementación de conjunto usando el DiccionarioPersonalizado
- Funcionalidad:
    - Almacena elementos únicos
    - Operaciones básicas de conjunto (agregar, __contains__, __iter__

Clases del Modelo (TDA)

Empresa

- Responsabilidad: Representa una empresa con puntos de atención
- Atributos clave:
    - id_empresa, nombre, abreviatura
    - puntos_atencion: Lista de puntos de atención

- transacciones: Lista de transacciones disponibles

PuntoAtencion

- Responsabilidad: Gestiona un punto físico de atención al cliente
- Atributos clave:
  - escritorios: Lista de escritorios disponibles
  - clientes_en_espera: Cola FIFO de clientes esperando
  - clientes_atendidos: Lista de clientes ya atendidos

EscritorioServicio

- Responsabilidad: Representa un escritorio de atención
- Atributos clave:
  - activo: Estado del escritorio
  - cliente_actual: Cliente siendo atendido
  - tiempo_restante: Tiempo faltante para terminar la atención actual

Transaccion

- Responsabilidad: Define un tipo de transacción con su tiempo de atención
- Atributos clave:
  - id_transaccion, nombre
  - tiempo_atencion: Tiempo que toma realizar la transacción

Cliente

- Responsabilidad: Representa a un cliente con sus transacciones
- Atributos clave:
  - dpi, nombre
  - transacciones: Lista de transacciones que necesita
  - tiempo_espera: Tiempo estimado de espera
  - ticket: Identificador único de atención

Sistema Principal

SistemaAtencion

- Responsabilidad: Coordinar toda la lógica del sistema
- Funcionalidades clave:
  - Carga de configuración desde XML
  - Gestión de empresas, puntos y escritorios
  - Asignación de clientes y simulación de atención
  - Generación de reportes estadísticos
  - Manejo de tickets únicos

Interfaz Gráfica

MobileAppSimulator

- Responsabilidad: Proporcionar interfaz de usuario para el sistema
- Componentes principales:
  - Menús para cargar configuración y generar reportes
  - Controles para seleccionar empresa/punto
  - Listado de transacciones disponibles
  - Gestión de escritorios (activar/desactivar)
  - Diálogos para ingresar datos de clientes

Descripción de Librerías Empleadas

- tkinter: Biblioteca estándar de Python para crear interfaces gráficas
  - Componentes usados: Frames, Labels, Buttons, Combobox, Menus
  - ttk para widgets temáticos modernos

- PIL (Python Imaging Library): Para manejo de imágenes
    - Usado en la visualización de reportes gráficos
- graphviz: Generación de gráficos/diagramas
    - Usado para crear reportes visuales del sistema
- xml.etree.ElementTree: Procesamiento de XML
    - Para cargar/salvar configuración del sistema
- random: Generación de números aleatorios
    - Para crear tickets únicos
- tempfile y os: Manejo de archivos temporales
    - Para guardar imágenes de reportes temporalmente

Conceptos de POO Aplicados

Encapsulación:

Todas las clases ocultan sus atributos internos y exponen métodos controlados

Ejemplo: ListaEnlazada maneja sus nodos internamente

Abstracción:

Las clases del modelo representan conceptos del dominio (Empresa, PuntoAtencion, etc.)

La interfaz gráfica abstrae la complejidad del sistema subyacente

Herencia:

MiString, MiNumero y MiCaracter heredan de MiObjeto

Permite compartir comportamiento común (comparaciones, verificaciones)

Polimorfismo:

El DiccionarioPersonalizado puede manejar diferentes tipos de claves

Métodos como igual_a() se implementan de forma específica en cada clase

Composición:

Empresa contiene PuntoAtencion, que a su vez contiene EscritorioServicio

SistemaAtencion compone todas las demás clases

Conclusiones

Estructuras de datos personalizadas: El proyecto demostró que es posible implementar estructuras complejas como diccionarios y conjuntos sin depender de las estructuras nativas de Python, aunque con un costo en rendimiento.

Manejo de estado: El sistema efectivamente modela el estado de múltiples empresas, puntos de atención y clientes, manteniendo consistencia durante las operaciones.

Interfaz gráfica: La interfaz basada en Tkinter, aunque simple, provee una forma efectiva de interactuar con el sistema subyacente.

Persistencia: El uso de XML para configuración permite guardar y cargar el estado del sistema de manera estructurada.

Simulación: La capacidad de avanzar el tiempo y simular la atención demuestra un modelo efectivo de colas y prioridades.

POO: El diseño orientado a objetos permitió organizar el código en componentes cohesivos y débilmente acoplados.

Apéndice

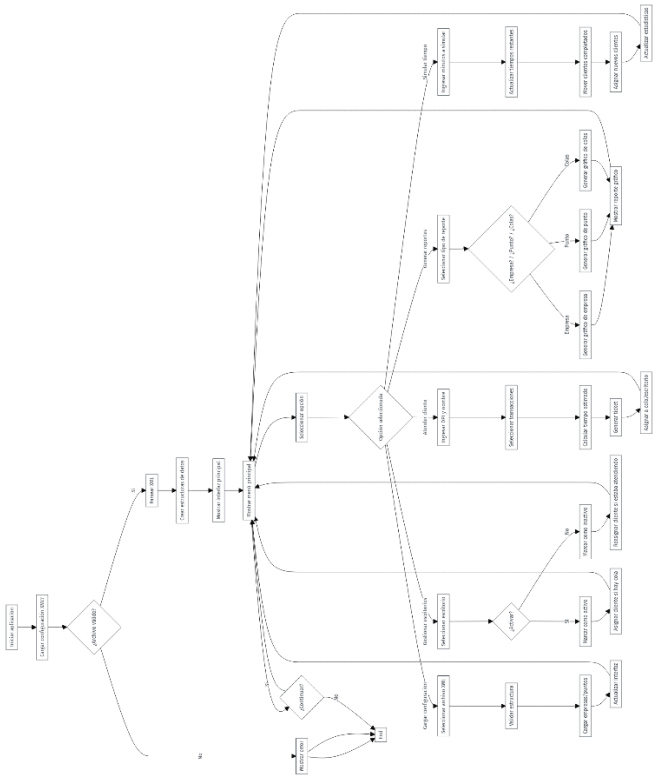

Figura 1. Diagrama de Actividades

Fuente: Elaboración propia (2025)

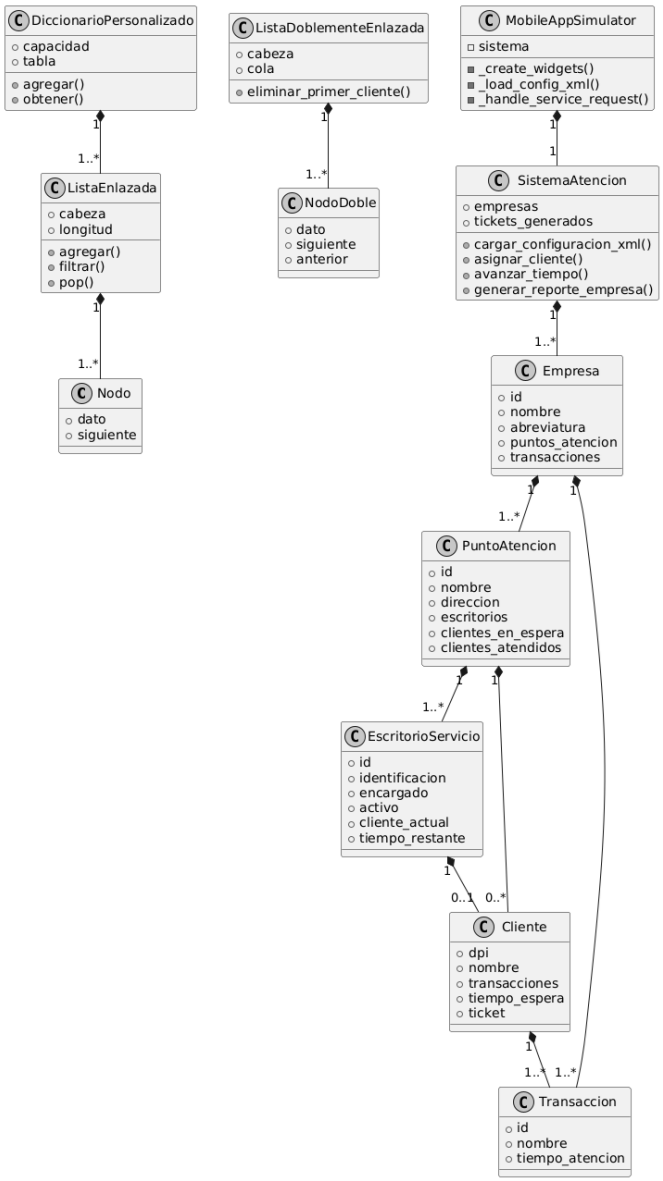

Figura 2. Diagrama de Clases

Fuente: Elaboración propia (2025)

Referencias Bibliográficas

Python Software Foundation. (2023). The Python Standard Library. Python Documentation. https://docs.python.org/3/library/

Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Graphviz Team. (2022). Graphviz - Graph Visualization Software. https://graphviz.org/

Python Pillow Developers. (2023). PIL/Pillow Documentation. https://pillow.readthedocs.io/

Tkinter Documentation. (2023). Tkinter 8.5 reference: a GUI for Python. https://docs.python.org/3/library/tkinter.html