DESAFÍO 2

Autores: *Maria Valentina Quiroga Alzate, Juan Felipe Orozco Londoño*

Informática II
Departamento de Ingeniería Electrónica y de Telecomunicaciones
Universidad de Antioquia

Abstract

Este documento presenta el análisis y diseño de un sistema de gestión de reservas de estadías tipo homestay, inspirado en plataformas digitales de alojamiento temporal. El sistema fue desarrollado en lenguaje C++ utilizando programación orientada a objetos, con restricciones que prohíben el uso de la STL (salvo string) y exigen estructuras de datos propias implementadas con memoria dinámica. El modelo se compone de cinco entidades principales: Usuario, Administrador, Alojamiento, Reservación y Fecha, permitiendo operaciones como reservas, cancelaciones, validación de solapamientos y actualización del histórico. La lógica está organizada mediante funciones auxiliares conectadas a un menú central, lo que da lugar a una solución modular, clara y eficiente.

Introducción

El desarrollo de aplicaciones que gestionen reservas de alojamientos se ha convertido en un escenario ideal para aplicar conceptos avanzados de programación orientada a objetos, especialmente en entornos educativos donde se busca reforzar la comprensión de estructuras dinámicas, relaciones entre clases y aprender a mejorar la eficiencia en el uso de memoria. En este contexto, el presente desafío propone implementar un sistema que permita gestionar estadías ofrecidas por anfitriones y reservadas por huéspedes, simulando el funcionamiento básico de plataformas como Airbnb. La solución requiere modelar y conectar entidades claves del dominio, tales como personas (huéspedes y anfitriones), unidades habitacionales (alojamientos) y acciones de reserva. Estas entidades deben interactuar entre sí respetando condiciones como la disponibilidad temporal, la ausencia de traslapes en reservas y la actualización constante de registros. A su vez, el sistema debe registrar la información de manera persistente en archivos de texto estructurados, y operar bajo un esquema de modularidad y eficiencia. Una de las decisiones centrales del diseño fue evitar la centralización del control en una sola clase administradora. En su lugar, se optó por una arquitectura distribuida, donde el flujo del sistema se gestiona desde funciones auxiliares organizadas y desde el main, interactuando con objetos concretos y estructuras dinámicas en memoria. Esto permite mantener una alta cohesión entre clases y una separación clara de responsabilidades, además de cumplir con la restricción explícita de no utilizar estructuras de la STL (con excepción de string).

Marco teórico

Modelamiento de sistemas en programación

El modelamiento de sistemas consiste en representar los elementos y relaciones que forman parte de un sistema antes de programarlo. Sirve

para entender cómo funcionará la solución, organizar mejor el código y detectar posibles errores desde el diseño. En programación orientada a objetos, esto se hace principalmente mediante diagramas de clases y definición de atributos y métodos por clase.

Programación orientada a objetos (POO)

La programación orientada a objetos es un estilo de programación que se basa en el uso de objetos que combinan datos (atributos) y funciones (métodos). Ayuda a estructurar mejor el código, hacerlo más reutilizable y fácil de mantener.

- Abstracción: Es el proceso de enfocarse solo en lo importante de un objeto y dejar de lado los detalles innecesarios. Por ejemplo, un objeto `Usuario` puede tener nombre y documento, sin importar cómo se almacena internamente.
- Encapsulación: Consiste en proteger los datos de una clase para que solo puedan ser modificados a través de métodos definidos, evitando errores y mejorando la seguridad del código.
- Diagrama de clases: Es un dibujo que muestra las clases de un sistema, sus atributos, métodos y cómo se relacionan entre ellas. Es útil para organizar el diseño antes de codificar.
- Funciones amigas: Son funciones externas a una clase que pueden acceder a sus atributos privados. Se usan cuando dos clases necesitan compartir información interna de forma controlada.
- Sobrecarga: Es la posibilidad de usar el mismo nombre para varias funciones u operadores, siempre que tengan diferentes parámetros. Permite escribir código más limpio y entendible.

Estrategia propuesta para la solución y Diseño lógico del modelo

La solución fue estructurada en torno a cinco clases principales: **Fecha**, **Reserva**, **Alojamiento**, **Usuario** y **Administrador**, cada una con una responsabilidad bien delimitada. El diseño se basó en una representación fiel del dominio, donde:

- **Fecha** encapsula el manejo de fechas: comparación, suma de noches y presentación en formato legible.
- **Reserva** representa una acción de alquiler entre un usuario y un alojamiento, identificados a través de sus respectivos códigos o documentos txt.
- **Alojamiento** almacena la información general del inmueble, su relación con un administrador y las reservas que tiene asociadas.
- **Usuario** mantiene el historial de reservas activas de un cliente y valida conflictos de fechas antes de aceptar una nueva reserva.
- **Administrador** gestiona uno o varios alojamientos, con la capacidad de visualizar las reservas asociadas a cada uno.

Las relaciones entre clases se establecen principalmente mediante **composición** (por ejemplo, un `Usuario` contiene un arreglo dinámico `Reserva**`) o **asociación vía identificador** (por ejemplo, mediante `codigoAlojamiento` o

`documento`). Esto permite reducir el acoplamiento entre clases y facilita un control total sobre la memoria sin duplicar información innecesaria.

Todo el sistema opera sobre **arreglos dinámicos gestionados manualmente con punteros dobles** (`Clase**`), permitiendo almacenar múltiples instancias de objetos sin utilizar vectores ni listas enlazadas de la STL. Se implementaron constructores parametrizados, constructores de copia y destructores personalizados para asegurar la correcta gestión y liberación de memoria en todas las clases que manejan estructuras dinámicas.

Archivos disponibles

Para la persistencia de la información, se utilizaron los siguientes archivos de texto:

1. `usuarios.txt` – contiene tanto a usuarios como a administradores, diferenciados mediante un campo de rol.

2. `alojamiento_y_administrador.txt` – almacena la información de los alojamientos junto con el documento del administrador responsable.

3. `reservas_vigentes.txt` – contiene las reservas activas cargadas en el sistema.

4. `historico.txt` – registra todas las reservas finalizadas o vencidas.

Consideraciones de desarrollo

Durante el desarrollo se va a evitar el uso de bibliotecas de la STL, en cumplimiento de las restricciones establecidas en el desafío, por lo que esto implicará la implementación manual de arreglos dinámicos utilizando punteros dobles, así como la gestión explícita de memoria en todas las clases que almacenen colecciones de objetos. Se propone utilizar tipos de datos simples como `char[]` para los identificadores, con el objetivo de reducir el uso de memoria y facilitar la lectura y escritura de datos en archivos de texto. Para campos de texto más extensos o variables, como direcciones o comentarios, se empleará el tipo `std::string`, ya que facilita su manipulación durante la ejecución. La lógica del sistema se organizará de modo que las operaciones principales (búsquedas, validaciones, registros) se realicen en memoria, y las actualizaciones a los archivos se efectúen únicamente cuando sea necesario, además, el control del sistema se llevará a cabo desde el archivo `main.cpp`, donde se implementará un menú interactivo que active funcionalidades específicas mediante funciones auxiliares que se van a encargar de interactuar con los objetos cargados en memoria, realizar búsquedas, validar condiciones del negocio y actualizar los archivos de texto de acuerdo con las acciones realizadas por el usuario.

Diagrama de clases UML

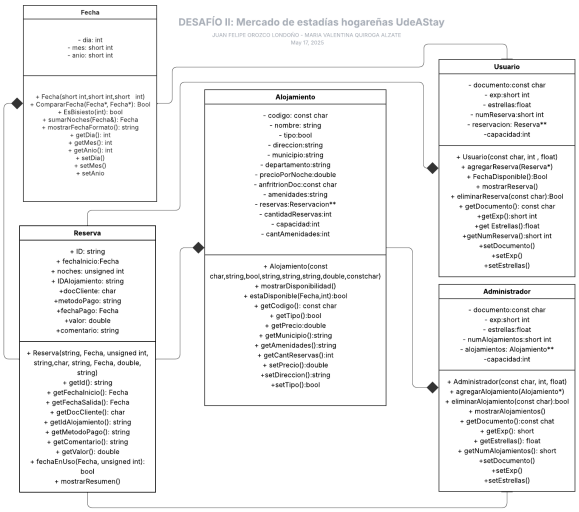

Figura 1: Diagrama de clases UML.

El diagrama de clases está constituido por cinco clases principales (Usuario, Administrador,

Alojamiento, Reserva y Fecha) conectadas mediante relaciones de asociación y composición, según la notación UML estándar. Las asociaciones, representadas con líneas rectas (por ejemplo, entre Usuario y Reserva), indican que una clase utiliza o se relaciona con otra sin que exista una dependencia vital entre sus ciclos de vida. Por ejemplo, un Usuario puede existir sin reservas, y una Reserva puede referenciar a un Alojamiento existente.

En cambio, la composición, representada con un rombo negro (como entre Reserva y Fecha), implica una relación de contención fuerte donde los objetos contenidos (Fecha) dependen completamente de la existencia del objeto principal (Reserva). Es decir, si una reserva se elimina, sus fechas asociadas también dejan de existir.

Conclusiónes

- El planteamiento de la solución permitió aplicar y entender mejor los conceptos de la programación orientada a objetos, como encapsulamiento, modularidad, abstracción y composición, a través de clases con responsabilidades bien definidas.
- Se propuso un diseño distribuido, evitando una clase controladora central. En su lugar, las clases como `Usuario`, `Administrador`, `Alojamiento`, `Reserva` y `Fecha` interactúan de manera independiente mediante identificadores, lo que mejora la cohesión y reduce el acoplamiento.
- Las relaciones entre objetos se modelaron usando composición o asociación por identificador, lo cual facilita la persistencia de datos en archivos de texto y mantiene el diseño liviano y fácil de mantener.