Informe del desafío 2

# Hugo Esteban Barrero García – Juan Angel Omaña Montañez

# Universidad de Antioquia

# Informática 2

# Aníbal José Guerra Soler – Augusto Salazar

# Medellín, Colombia

Mayo, 2025

**1. Introducción**

El presente informe acompaña la primera entrega del Desafío II de la asignatura Informática II, correspondiente al diseño del **Diagrama de Clases UML** para el sistema denominado **UdeAStay**. Este sistema busca representar una plataforma de gestión de reservas de alojamiento entre anfitriones y huéspedes, modelando de manera clara las entidades involucradas, sus atributos, métodos y relaciones.

**2. Objetivo del diseño**

El objetivo principal fue desarrollar un modelo de clases que permita:

* Gestionar alojamientos ofrecidos por anfitriones.
* Permitir que huéspedes realicen reservas.
* Registrar información detallada sobre cada reservación.
* Validar disponibilidad y fechas.
* Controlar estructuras dinámicas de datos, sin uso de STL ni herencia.

El diagrama se ha elaborado en concordancia con las restricciones del enunciado: no uso de herencia, buena modularidad, manejo dinámico de memoria, encapsulamiento, y uso eficiente de tipos de datos.

**3. Clases implementadas**

Se definieron seis clases principales, cada una representando una entidad central del sistema:

* **Fecha**: utilizada para representar fechas de entrada, pago y corte. Se conecta con Reservacion y UdeAStay.
* **Reservacion**: modela las reservas hechas por los huéspedes. Se conecta a Huesped, Alojamiento y Fecha.
* **Huesped**: representa a los usuarios que realizan reservas. Se conecta a Reservacion.
* **Alojamiento**: representa los espacios ofrecidos. Se conecta a Reservacion y Anfitrion.
* **Anfitrion**: representa al usuario que administra alojamientos. Se conecta directamente con Alojamiento.
* **UdeAStay**: clase principal del sistema. Administra las listas de todas las entidades anteriores y controla la lógica general del programa.

**4. Relaciones entre clases**

El diseño considera únicamente relaciones necesarias y funcionales, respetando principios de cohesión y bajo acoplamiento:

* UdeAStay se conecta con todas las clases porque gestiona directamente las listas dinámicas.
* Reservacion se conecta a Huesped y Alojamiento, además de usar Fecha para los campos fechaEntrada y fechaPago.
* Alojamiento está vinculado a un único Anfitrion, lo que permite acceder a las reservas del anfitrión a través de sus alojamientos.

**5. Implementación de clases**

A continuación se detallan las decisiones clave tomadas en la implementación de las primeras clases:

**5.1 Clase Fecha**

Se diseñó como una clase sencilla para representar días, meses y años, sin uso de memoria dinámica. Se mejoró con validación de fechas en el constructor y setters para evitar valores inválidos. Se implementaron operadores de comparación (==, <, <=, >=), el operador de suma (+) para calcular fechas futuras, y el operador de asignación = para garantizar copias correctas. El método toString() genera una representación textual dinámica de la fecha.

**5.2 Clase Anfitrion**

Modela al usuario dueño de alojamientos. Maneja un arreglo dinámico de punteros a alojamientos, y permite agregar nuevos alojamientos, ver reservas dentro de un rango, y anularlas de forma simulada.  
La memoria dinámica se gestiona con redimensionamiento progresivo para mantener eficiencia. El uso de punteros favorece la referencia cruzada entre anfitrión y alojamiento sin duplicar datos.

**5.3 Clase Alojamiento**

Representa los lugares ofertados por los anfitriones. Contiene información textual y numérica del espacio, su precio, tipo y amenidades. Estas últimas se gestionan mediante una codificación bitmask, que permite activar o desactivar múltiples características de forma eficiente.  
Cada alojamiento mantiene su propio historial de fechas reservadas, y permite verificar disponibilidad, registrar nuevas reservas, y mostrar las que se encuentren dentro de un intervalo.  
Se incluyeron métodos privados como copiarTexto y redimensionarReservas para mantener encapsulamiento, y se mejoró la seguridad al reemplazar funciones peligrosas como strcpy por strncpy.

**5.4 Clase Reserva**

Encapsula toda la información relacionada con una reserva, incluyendo código único generado automáticamente (RSV-0001, etc.), fechas de entrada y pago, duración, alojamiento reservado, huésped responsable, medio de pago, monto pagado y anotaciones opcionales. Se implementaron dos constructores (uno automático y otro con código manual para carga desde archivos), copia profunda, operador de asignación, y validaciones internas. Además, cuenta con métodos para obtener la fecha de salida, imprimir comprobantes con formato detallado, y acceder de forma segura a todos sus atributos.

**5.5 Clase Huesped**

Representa a los usuarios que realizan reservas. Maneja nombre completo, documento, antigüedad, puntuación y una lista dinámica de reservas. El código del huésped se genera automáticamente (HSP-0001, etc.). Incluye métodos para agregar reservas (validando que no se solapen), anular reservas (notificando al alojamiento), imprimir comprobantes de todas las reservas y mostrar un resumen del estado del huésped.

**6. Pruebas realizadas**

Se desarrollaron pruebas específicas para cada clase, verificando su comportamiento esperado:

* En Fecha se probaron comparaciones, copias, conversiones a texto y operadores personalizados.
* En Alojamiento se verificó la gestión de amenidades, el registro de reservas, la disponibilidad por fechas y la impresión de datos.
* En Anfitrion se comprobó la correcta asociación con alojamientos y la capacidad de consultar y anular reservas.
* **En Reserva** se validó la creación de reservas mediante ambos constructores, copia, asignación, generación de códigos, validación de fechas y la impresión del comprobante completo.
* En **Huesped** se comprobó la gestión dinámica de reservas, prevención de solapamientos, anulación correcta con referencia al alojamiento, generación automática de código y resumen de estado del huésped.  
  Durante las pruebas de integración de las clases Huesped y Reserva, se logró validar completamente su funcionamiento dentro del sistema. Se verificaron casos como la creación automática de códigos, el almacenamiento y recuperación de reservas, la detección de conflictos de fechas al agregar nuevas reservas, y la correcta anulación de reservas, incluyendo la liberación de memoria dinámica.
* También se validaron los contadores de recursos incorporados (totalHuespedesCreados, totalReservasCreadas, totalIteraciones...) que permiten analizar la eficiencia del sistema sin herramientas externas. Las impresiones de comprobantes y los resúmenes de información se mostraron correctamente, sin errores en tiempo de ejecución ni fugas de memoria observadas.
* Esta versión se diseñó evitando completamente dependencias de librerías como <cstring> o <iostream>, reemplazándolas por funciones auxiliares propias para copiar, comparar y gestionar texto. Esto refuerza la independencia del sistema, permitiendo mayor control sobre memoria, ciclos y recursos.  
  Durante el desarrollo del sistema *UdeAStay*, se presentaron diversos retos técnicos asociados al manejo de memoria dinámica, validaciones de entrada y consistencia entre clases. Uno de los principales problemas fue la duplicación de funciones auxiliares como copiarTexto, sonIguales, iniciaCon y longitudTexto, las cuales estaban repetidas en clases como Anfitrion, Huesped, Alojamiento, Reserva y UdeAStay. Esto generaba código innecesariamente largo, difícil de mantener y propenso a errores. Para solucionarlo, se creó un archivo central llamado Funciones.h donde se unificaron dichas funciones, y todas las clases se adaptaron para utilizar esta única fuente, asegurando consistencia y facilitando su depuración.
* También se identificaron errores en el manejo de memoria dinámica, especialmente en constructores de copia y operadores de asignación, donde se estaban realizando copias superficiales o se omitía la liberación de memoria anterior, lo cual podía provocar **fugas de memoria** y comportamientos impredecibles. Se corrigieron todos estos puntos implementando copias profundas correctamente y asegurando que los destructores eliminaran todas las referencias dinámicas asignadas.
* En cuanto a la interacción con el usuario, inicialmente el programa se quedaba bloqueado si el usuario ingresaba texto en lugar de números, especialmente en fechas o cantidades, debido a la falta de validación en las entradas. Para evitar esto, se mejoró el main.cpp incorporando validaciones robustas mediante cin.fail(), cin.clear() y cin.ignore(), garantizando que el sistema nunca se congele ni permita entradas corruptas. Además, se añadieron mensajes explicativos ante errores, lo cual mejora la experiencia del usuario.
* En la lógica de negocio, se implementó un sistema de **detección de conflictos de fechas** en las reservas: si un huésped intentaba reservar un alojamiento en una fecha que ya se encontraba ocupada, el sistema detecta el solapamiento y lo impide. Asimismo, se refactorizaron métodos como anularReservacion para que funcionaran por fecha y duración en lugar de por código, haciendo el sistema más flexible.
* Finalmente, se ejecutaron pruebas funcionales exigentes: se cargaron archivos .txt con estructuras complejas, se intentaron reservas en fechas no disponibles, se validaron anulaciones, consultas por anfitrión, y actualizaciones del histórico. El sistema también fue evaluado con entradas inválidas, como letras en campos numéricos o fechas mal formateadas, y todas las respuestas fueron correctas. Las métricas de consumo interno —como iteraciones, memoria y archivos abiertos— también fueron revisadas, confirmando que el sistema es estable, eficiente y listo para futuras ampliaciones.

### **10. Bibliografía y Referencias**

* Documentación oficial de Qt. Disponible en: <https://doc.qt.io>
* Apuntes de clase de la asignatura Informática II.
* Archivo entregado “Desafio Info II 2025-1 v1.01.pdf” con instrucciones del reto.
* Explicaciones técnicas desarrolladas durante el trabajo práctico.
* Discusiones y ayudas recibidas a través del canal del curso.
* Asistencia generada con ChatGPT (OpenAI).

Repositorio del proyecto: [https://github.com/Hugo-24/Desafio-2](https://github.com/Hugo-24/Desafio-21).