# **Informe Desafio II UdeA Stay**

## **Introducción**

En el presente informe daré una explicación en alto nivel del análisis e implementación del problema propuesto por los profesores “Desafío II”, se hablará sobre el diagrama de clase y las clases, además de las funciones y funcionalidades implementadas todo esto desde un alto nivel o en prosa.

La solución planteada surge por la necesidad que hallamos en que cada clase y sus diversas instancias no puede estar dentro de otra y por eso llegamos a la conclusión de las 5 clases.

la decisión de los módulos recae en una mejor lectura y separación coherente de las diferentes funciones y funcionalidades.

El txt tuvo una elección tanto humana como técnica, humana en el sentido de que se entiende lo que se lee y técnica en el sentido de que es la mejor que encontramos,todo se reduce a dos archivos alojamientos y reservas, porque solo dos?, porque en reserva es coherente y eficiente encapsular toda la información del huésped y de la reserva misma eso facilitando mejorando la lectura y asignación de los objetos huésped y reserva, para el caso alojamiento es similar es lógico encapsular a un alojamiento y su anfitrión, lo cual tanto potencia como facilita la lectura,interpretación y asignación de la información

## **Descripción de Clases**

### **Clase Alojamiento**

Encapsula la información de un alojamiento como nombre, código, documento del anfitrión, puntuación, tipo (casa o apartamento), ubicación, dirección, precio, y amenidades. Además, mantiene un arreglo de punteros a reservas y un contador. Incluye constructores, getters, setters y métodos para comprobar disponibilidad, aplicar filtros, imprimir información, cancelar reservas y actualizar el historial.

### **Clase Anfitrión**

Representa al dueño del alojamiento, incluyendo puntuación, antigüedad, documento de identidad y los alojamientos asociados. Dispone de métodos para consultar reservas, interactuar mediante un menú, y modificar o acceder a sus atributos.

### **Clase Fecha**

Administra fechas mediante atributos día, mes y año. Incluye validaciones, obtención del nombre del día (Zeller), comprobación de años bisiestos, suma de noches, y sobrecargas para comparaciones. Implementa métodos para mostrar la fecha y acceder a sus componentes.

### **Clase Huésped**

Contiene datos como puntuación, antigüedad y documento. Mantiene un arreglo de punteros a reservas donde se encuentran sus reservas pero en manera de direccion para evitar copias innecesaria en memoria. Ofrece funcionalidades para gestionar reservas, verificar disponibilidad, cancelar, actualizar y navegar mediante menú interactivo.

### **Clase Reserva**

Vincula a un huésped a un alojamiento en una fecha específica por medio de un puntero a su documento lo cual como en huésped nos evita crear copias innecesarias. Incluye campos como código, duración en noches, método de pago, fecha de pago, monto y comentario. Dispone de métodos para verificación de fechas, cálculo de valor, generación de comprobantes y acceso/modificación de sus campos.

## **Interacción entre Clases**

* La clase Reserva se relaciona tanto con Huésped como con Alojamiento mediante los atributos documento y codigoA, respectivamente.
* La clase Huésped y Anfitrión almacenan arreglos de punteros a reservas o alojamientos, mostrando una relación de agregación.
* Las funciones externas permiten manipular estas relaciones cargando, asignando y actualizando datos entre objetos de distintas clases.
* NOTA: por comodidad del lector y facilidad el diagrama de clase se encuentra al final

## **Descripción de Funciones Principales**

### **Función reserva**

void reserva(Alojamiento\* alojamientos, Reserva\*\* reservas, Huesped\* huespedes,

unsigned int &tamano1, unsigned int &tamano2, unsigned int &tamanoH, string &documento, unsigned int &contadoriteraciones);

Permite a un huésped generar una nueva reserva. Recibe los arreglos de alojamientos, reservas e información del huésped, junto con sus tamaños respectivos y el documento de identidad del huésped. Utiliza un contador de iteraciones para rastrear procesos internos.

### **Función cancelareserva**

void cancelareserva(Huesped \*huespedes, Reserva \*\*reservas, Alojamiento\* alojamientos, unsigned int &conthu, unsigned int &contA,

unsigned int &contR, string &codigo, unsigned int &contador);

Cancela una reserva específica a partir de su código. Afecta tanto al huésped como al alojamiento y elimina la reserva del sistema, actualizando los respectivos contadores.

### **Función usofiltro**

bool usofiltro(Alojamiento& alojamientos,unsigned int precio,float puntuacion,unsigned short int decision,unsigned int &contador);

Evalúa si un alojamiento cumple con un filtro específico según precio, puntuación o ambos. Retorna un valor booleano según el resultado del filtro.

### **Función actualizarHistorico**

void actualizarHistorico(Reserva \*\*reservas, unsigned int &tamano, unsigned int &contador);

Se encarga de tomar una fecha de “corte” y en base a la misma ir “eliminado/liberando” del arreglo y sus diferentes relaciones las reservas que cumplan la condicion, para a posteriori agregarlas a un purmanente(txt) tambien para mejorar la eficiencia en el uso de memoria “redimensiona” los arreglos implicados por medio de la fucnion auxiliar, void compactarReservas(Reserva\*\* reservas, unsigned int& tamano, unsigned int &contador);

## **Funciones Auxiliares (Descripción General)**

Estas funciones cumplen tareas auxiliares como inicialización, carga de datos, conversión de strings, validación sintáctica y lógica de municipios, y asignaciones básicas. No se detallan dado a su poca complejidad y su labor es ser auxiliares para verificaciones o tareas muy puntuales:

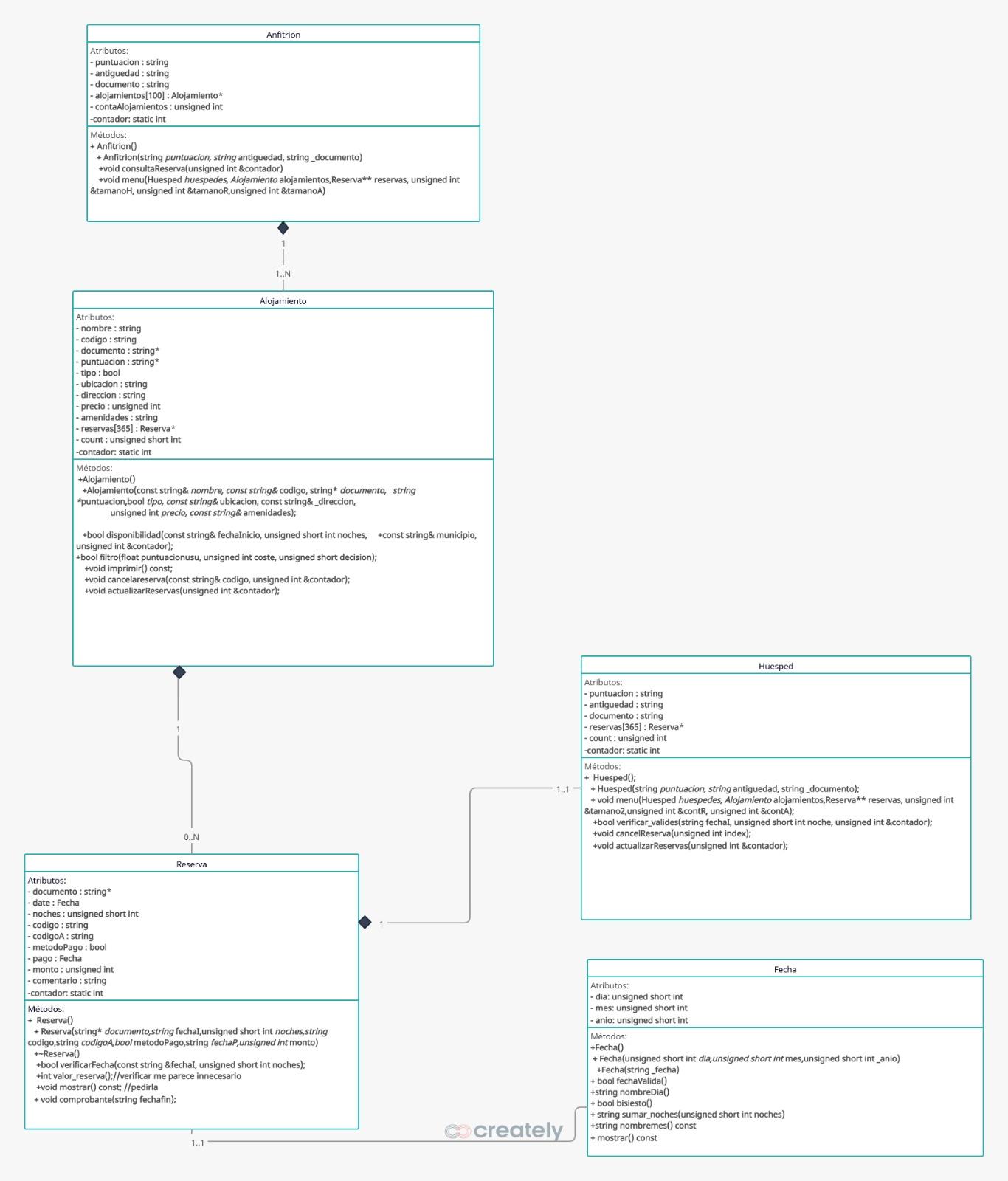
* actualizarpermanentereserva
* ingresar\_sistema
* cantidadLineas
* crearAnfitriones
* crearHuespedes
* asignarReservasA
* asignarReservah
* obtenerFechaActual
* aMinuscula
* Sintaxisvalida
* municipioexiste
* huespedDisponible
* codigoR
* comentario

## **Conclusión**

El diseño del sistema prioriza la gestión eficiente de objetos mediante punteros, facilitando la interacción entre clases sin duplicar datos, lo que optimiza el uso de memoria y permite una estructura modular y escalable.

PROBLEMAS EN EJECUCIÓN

La constante verificación de cada parámetro ingresado por el usuario genera muchos conflictos a la hora de probar y de contar iteraciones además toca hacer muchas estructuras adicionales.



Fecha

bool operator<(const Fecha& otra)const;

bool operator>(const Fecha& otra) const;