Programación 4

EXAMEN DICIEMBRE 2023

Por favor siga las siguientes indicaciones:

- Escriba con lápiz y de un solo lado de las hojas
- Escriba su nombre y número de documento en todas las hojas que entregue
- Numere las hojas e indique el total de hojas en la primera de ellas
- Recuerde entregar su número de examen
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el examen

Problema 1 (30 puntos)

Se desea crear un software para registrar las consultas y turnos de médicos de un prestador de salud. Los médicos se registran con un número de médico (único), nombre y están asociados a una o más especialidades, de las cuales se conoce su nombre. Un médico tiene agendado turnos en uno o más centros. Un centro es una locación física con nombre que lo identifica (por ej., Hospital Bermúdez) y una dirección. Cada turno especifica, para un día dado (por ej., 23 de diciembre), en qué centro atenderá un médico. Un médico no atiende en más de un centro un mismo día, pero si pueden atender varios médicos en un mismo centro un mismo día sin importar sus especialidades. Los pacientes se registran por un número de cédula (que los identifica), un nombre y un número de teléfono. Una consulta se crea para indicar que un paciente se atenderá en un turno en particular. Cada consulta además tiene un costo en pesos uruguayos, el cual es diferente en cada caso. Para registrarse con una consulta a un médico un paciente debe tener un pase vigente a una especialidad del médico a quien quiere consultar. Un pase es un registro que habilita a un paciente a consultar una especialidad particular. Los pases tienen una fecha de realizados y vigencia de un año.

Además, se cuenta con la descripción del siguiente Caso de Uso:

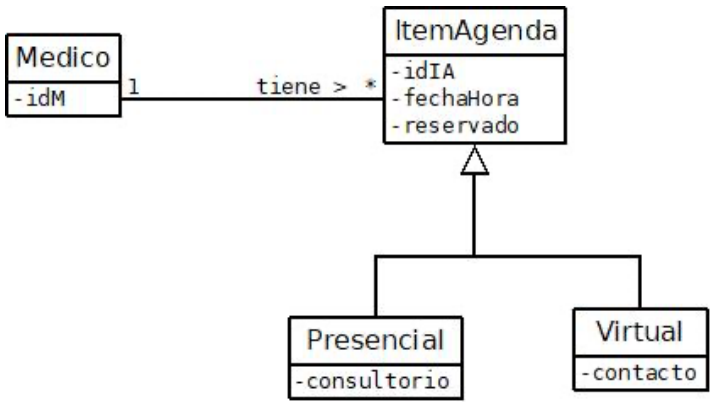
Caso de Uso	Agendar consulta
Actor	Usuario
Descripción	El caso de uso comienza cuando un usuario desea agendar la consulta de un paciente a un médico. Primero, el usuario ingresa el número de cédula del paciente. El Sistema indica la lista de especialidades para las cuales el paciente puede agendarse. Luego, el usuario entonces elige una especialidad indicando su nombre y el Sistema lista los primeros 30 turnos disponibles (o sea, nombre de médico, fecha, y nombre del centro) de esa especialidad. Si ningún turno le parece adecuado, puede solicitar al Sistema que liste los siguientes 30 turnos disponibles (y así sucesivamente hasta encontrar un turno o cancelar todo el caso de uso). Si en el algún momento un turno les parece adecuado al usuario, lo elige indicando nombre del médico, fecha y nombre del centro. Finalmente, el Sistema confirma la consulta seleccionada.

Se pide:

a) Realizar el Modelo de Dominio de la realidad planteada, incluyendo todas las restricciones que considere necesarias en lenguaje natural.
b) Realizar un Diagrama de Secuencia del Sistema (DSS) para el Caso de Uso. Indique el uso de memoria del Sistema y de datatypes, si corresponde.

Problema 2 (35 puntos)

La figura muestra el modelo de dominio de parte de un sistema de reserva de consultas médicas. Cada médico pone a disposición un conjunto de ítems de agenda que posteriormente se podrán utilizar para hacer consultas y reservas. Cada ítem de agenda tiene una fecha/hora, un identificador (único en el sistema) y una marca para indicar si está reservado.



Por otro lado, se tiene la especificación de las siguientes operaciones del sistema:

`agregarItemAgenda(idM : String, dataIA : DTDataItemAgenda)`	
Descripción	Agrega un ítem de agenda para un médico.
Parámetros	`idM`: identificador del médico `dataIA`: datos del ítem de agenda a agregar
Precondiciones	• Existe en el sistema una instancia `m:Medico` con `m.idM = idM` • No existe en el sistema una instancia `ia:ItemAgenda` con `ia.fechaHora = dataIA.fechaHora`
Postcondiciones	• Se crea una instancia `ia:ItemAgenda` con o `ia.fechaHora = dataIA.fechaHora` o `ia.idIA = Utils::nuevoIdIA()` o `ia.reservado = false` • Si `dataIA` es de tipo `DTDataPresencial`, entonces `ia` es de tipo `Presencial` y `ia.consultorio = dataIA.consultorio` • Si `dataIA` es de tipo `DTDataVirtual`, entonces `ia` es de tipo `Virtual` y `ia.contacto = dataIA.contacto` • Se crea un link entre la instancia `ia:ItemAgenda` y la instancia `m:Medico`

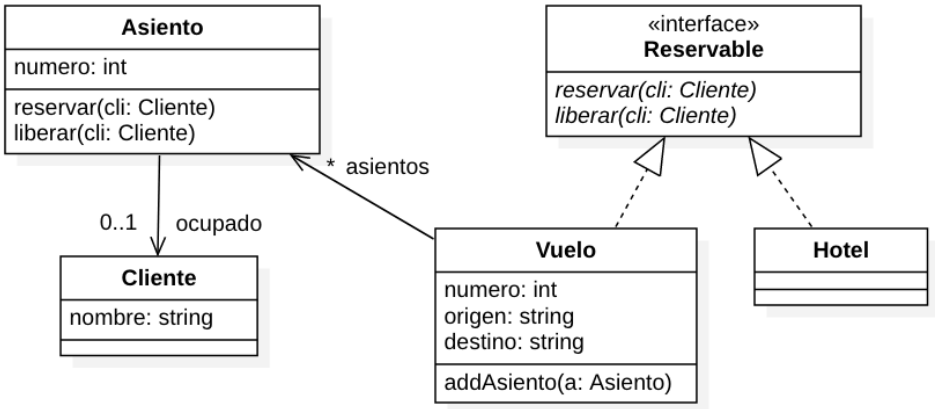
`buscarItemsDisponibles(idM : String):Set(DTDataItemAgenda)`	
Descripción	Devuelve los ítems de agenda disponibles para un médico dado.
Parámetros	`idM`: identificador del médico
Precondiciones	• Existe en el sistema una instancia `m:Medico` con `m.idM = idM.`
Postcondiciones	• Se retorna un conjunto de `DTDataItemAgenda` correspondiente a las instancias `ia:ItemAgenda` tales que o `ia` está asociada a la instancia `m:Medico` o `ia.reservado = false`.

Se pide:

a) Definir completamente el datatype `DTDataItemAgenda`.
b) Realizar los Diagramas de Comunicación de las dos operaciones especificadas.
c) Realizar el Diagrama de Clases de Diseño completo resultante. No incorporar setters, getters ni operaciones de colecciones, e incorporar sólo los constructores y destructores que se utilicen en los diagramas de comunicación.

Problema 3 (35 puntos)

Se está desarrollando un sistema de reservas turísticas que permite reservar diferentes elementos "reservables" como hoteles y vuelos. Se realizó el diseño de clases parcial correspondiente a los vuelos. Se conoce información del vuelo y sus asientos, así como si los asientos están o no ocupados/reservados por un cliente (que puede reservar varios asientos).



A continuación, se describen el comportamiento de las principales operaciones definidas.

`bool Vuelo::reservar(cli : Cliente)`
Ubica al cliente `cli` en un asiento disponible del vuelo y retorna `true`. En caso de que no sea posible hacerlo porque no hay asientos disponibles, retorna `false`.

`bool Asiento::reservar(cli : Cliente)`
Si el asiento no está ocupado, lo ocupa con el cliente `cli` y retorna `true`, o `false` en caso contrario.

`void Vuelo::liberar(cli : Cliente)`
Libera todos los asientos que se hayan reservado para el cliente `cli`.

`void Asiento::liberar(cli : Cliente)`
Libera el asiento en caso de que esté ocupado por el cliente `cli`.

Se pide:

a) Implementar en C++ el .h de la interfaz `Reservable`.
b) Implementar en C++ el .h de la clase `Vuelo`.
c) Implementar en C++ el .cc de la clase `Vuelo`.
d) Implementar en C++ el .h de la clase `Asiento`.
e) Implementar en C++ el .cc de la clase `Asiento`.
f) Implementar en C++ un main que permita
    a. Crear un cliente "Cliente"
    b. Crear el vuelo 101 de "Ciudad A" a "Ciudad B" con dos asientos (1 y 2)
    c. Reservar un asiento para el cliente y liberar dicha reserva para ese cliente

Observaciones:

- NO se pueden agregar más operaciones de las definidas en el diseño anterior.
- Utilice colecciones paramétricas (contenedores STL) y la clase std::string.
- No incluya directivas al precompilador (#include, etc).
- En los `.h` y `.cc`, incluir los constructores que sean necesarios. No incluir destructor ni operaciones get/set (puede asumir su existencia si se requieren en algún método)
- La implementación debe hacer un correcto manejo de memoria.