Programación 4

EXAMEN DICIEMBRE 2021

Problema 1 (30 puntos)

Se desea modelar un sistema de gestión de pasajes de avión para una conocida aerolínea. Existen dos tipos de usuarios (mutuamente excluyentes): aquellos que usarán el sistema vía web y los funcionarios de la aerolínea que lo usarán en sus oficinas. Todos tendrán un nick (que los identifica), un nombre y un número de teléfono. Además, ambos tipos de usuario pueden hacer reservas de pasajes de avión. Las reservas se realizan sobre los vuelos, que son entre dos ciudades. Los vuelos tienen un número que los identifica, una ciudad de origen y una de destino (diferentes entre sí) y una fecha en la cual se realizarán. Todas las ciudades tienen un nombre que las identifica y por lo menos un aeropuerto, del cual se conoce también su nombre. Cada reserva está asociada a un vuelo específico, y tiene un número de reserva que la identifica, un número de asiento y un monto a pagar. Un vuelo tiene una secuencia ordenada (según el número de asiento) de reservas. El pasajero de una reserva de la web siempre es el usuario web que la creó, mientras que una reserva creada por un funcionario puede estar a nombre de cualquier usuario web, aunque también se registra qué funcionario la creó. Una reserva de pasaje puede estar 'Pendiente', 'Pagada', 'Realizada' (el pasajero realizó el viaje), o 'Vencida' (el pasajero no hizo el viaje, aunque lo pagó).

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

Caso de Uso	Marcar reservas de pasajes como Pagadas
Actor	Funcionario aerolínea
Descripción	El caso de uso comienza cuando un funcionario que tiene una sesión iniciada en el sistema quiere marcar una o más reservas como pagadas. Para esto, el funcionario ingresa un número de vuelo. El sistema entonces lista todas las reservas de pasajes de ese vuelo que están pendientes. Para cada reserva se lista: número de asiento, nombre y número de teléfono del pasajero. El sistema solicita el número de la reserva a marcar como pagada. Luego de que el funcionario lo ingresa se pide confirmación (Si o No) y en caso afirmativo se confirma el cambio. Se le pregunta también al funcionario si se desea marcar más reservas pagas del mismo vuelo, en caso afirmativo se muestra la lista actualizada de reservas no pagadas de ese vuelo y se repite el proceso. Cuando se termina de trabajar con ese vuelo, el sistema pregunta si se desea trabajar con otro vuelo y en caso afirmativo se solicita nuevamente un número de vuelo y se repite el proceso. En caso negativo se da por finalizado el caso de uso.

Se pide:

a) Realizar el Modelo de Dominio de la realidad planteada, incluyendo restricciones en lenguaje natural.
b) Realizar un Diagrama de Secuencia del Sistema (DSS) para el Caso de Uso "Marcar reservas de pasajes como Pagadas" indicando el uso de memoria del Sistema y de datatypes si corresponde.

Programación 4

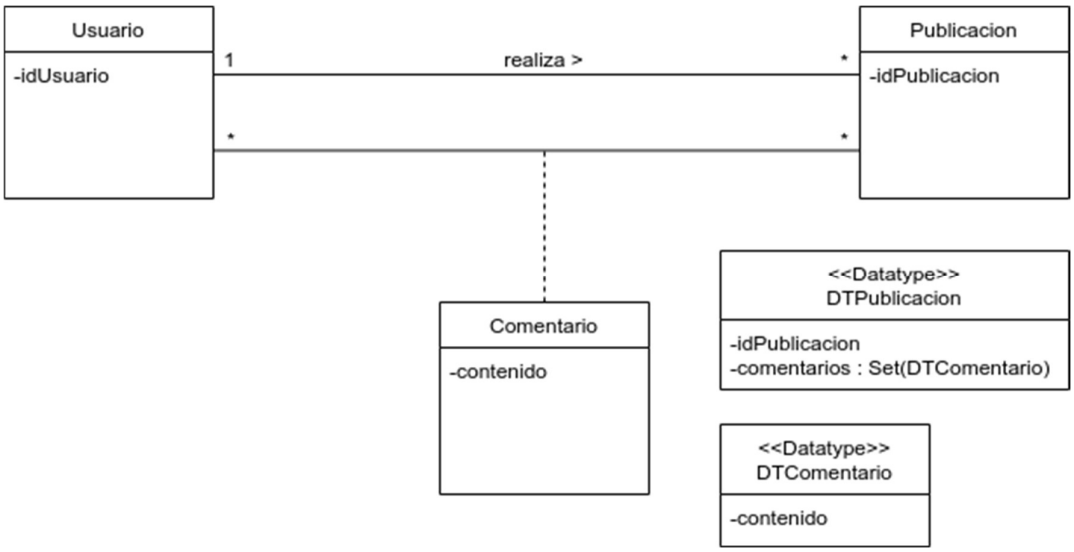
EXAMEN DICIEMBRE 2021

Problema 2 (35 puntos)

Un grupo de *fingers* (estudiantes de Fing) se encuentra desarrollando una nueva red social para la comunidad de facultad. Inicialmente en ella los/as usuarios/as podrán realizar diferentes publicaciones y también comentarios sobre las publicaciones propias o ajenas.

Parte I:

Considere el siguiente modelo de dominio del sistema de la red social, donde `idUsuario` identifica al `Usuario` e `idPublicacion` identifica a la `Publicacion`.



Por otra parte, se cuenta con los contratos de las siguientes operaciones del sistema:

`realizarComentario(idU : int, idP : int, texto : string)`	
Descripción:	Ingresa el comentario de un usuario en una publicación.
Parámetros:	`idU`: Identificador del usuario. `idP`: Identificador de la publicación. `texto`: Contenido del comentario.
Precondiciones:	Existe en el sistema un `Usuario` con atributo `idUsuario` igual a `idU`. Existe en el sistema una `Publicacion` con atributo `idPublicacion` igual a `idP`.
Poscondiciones:	Se crea una instancia `c` de `Comentario` con atributo `contenido` igual al parámetro `texto`. Se crea un link entre `c` y el `Usuario` identificado por `idU` y otro entre `c` y la `Publicacion` identificada por `idP`.

`listarPublicaciones(idU : int): Set(DTPublicacion)`	
Descripción:	Lista las publicaciones realizadas por un usuario. Para cada una de ellas se muestra su identificador y el contenido de los comentarios asociados.
Parámetros:	`idU`: Identificador del usuario.
Precondiciones:	Existe en el sistema un `Usuario` con atributo `idUsuario` igual a `idU`.
Poscondiciones:	Se crea un conjunto de instancias `DTPublicacion`, donde cada elemento corresponde a una instancia de `Publicacion` asociada al `Usuario`, e incluye el `idPublicacion` y el conjunto de `DTComentario`. Estos a su vez se corresponden con instancias de `Comentario` e incluyen el `contenido`.

Se pide:

a) Realizar los Diagramas de Comunicación correspondientes a las operaciones `realizarComentario` y `listarPublicaciones`, incluyendo visibilidades.
b) Realizar el Diagrama de Clases de Diseño resultante, incluyendo controladores y datatypes si corresponde. No incorporar setters ni getters. Incorporar sólo los constructores y destructores que se utilicen en los diagramas de comunicación.

Parte II:

En una siguiente fase del desarrollo se quiere ampliar las funcionalidades referentes a las publicaciones. Se sabe que habrá tres tipos de publicaciones (imagen, video y álbum) pero se espera que a futuro haya otros. Un álbum es un conjunto de otras publicaciones. Por otra parte, la operación `realizarPublicacion()`, perteneciente a la clase `Publicacion`, es la que se encarga de dejar disponible en la red social las publicaciones de todos los tipos.

Se pide:

c) Realizar el Diagrama de Clases de Diseño para modelar la situación descrita anteriormente, utilizando un patrón de diseño. Indicar el nombre del patrón, los roles de las clases participantes y los seudocódigos de las operaciones del patrón. Asuma que dispone de las siguientes operaciones:
   - `Video::publicarVideo()`: Deja disponible en la red social una publicación de tipo video.
   - `Imagen::publicarImagen()`: Deja disponible en la red social una publicación de tipo imagen.

Programación 4

EXAMEN DICIEMBRE 2021

Problema 3 (35 puntos)

Usted ha sido contratado por una empresa de desarrollo de software encargada de implementar un sistema para la gestión de un pequeño restaurante.

El restaurante cuenta con dos tipos de empleados: los Delivery que llevan pedidos a domicilio y los Mozos que atienden las mesas en el local. Los pedidos tienen dos posibles estados: "preparándose", que es el estado con el que ingresan al sistema y están hasta que el Delivery los retira, y "en camino", que es el estado a partir de ese momento. Por otro lado, las mesas también tienen dos estados que dependen de si tienen o no clientes en el momento. Por último, es de interés ser capaz de calcular todo lo facturado en una mesa en un día. Para esto la mesa mantiene un subtotal de todos los clientes que ya se retiraron y en caso de tener clientes al momento de facturar suma lo gastado hasta el momento. En la figura 1 puede ver el Diagrama de Clases de Diseño creado para el sistema.

Los responsables del proyecto no están seguros de que el patrón State aplique en los dos casos planteados, pero dado que hace mucho que cursaron Programación 4 lo consultan a usted para definir qué hacer con cada uno de ellos.

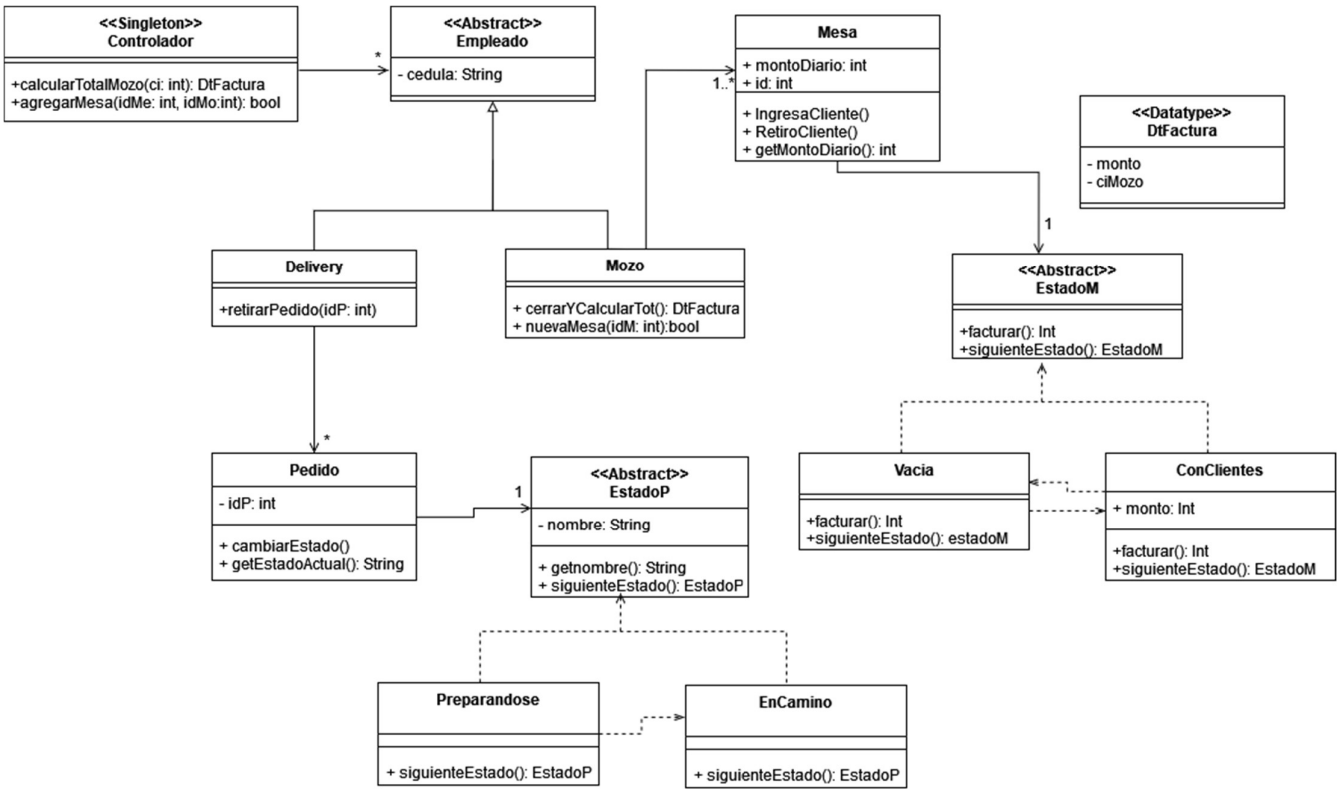


Figura 1

Se pide:

a) Explique brevemente, para cada aplicación del patrón State del diagrama, si es justificada o no y por qué.
b) Independientemente de lo contestado en la parte a) implemente la(s) operación(es) de las clases `Mozo` y `Mesa` que aparece(n) en el diagrama de comunicación de la figura 2.
c) Independientemente de lo contestado en la parte a) implemente los `.h` y `.cpp` correspondientes a las clases `EstadoM` y `ConClientes`. Implemente además el módulo `Controlador.h` y las operaciones `agregarMesa(…)` y `nuevaMesa(…)` de `Controlador.cpp` y `Mozo.cpp` respectivamente.

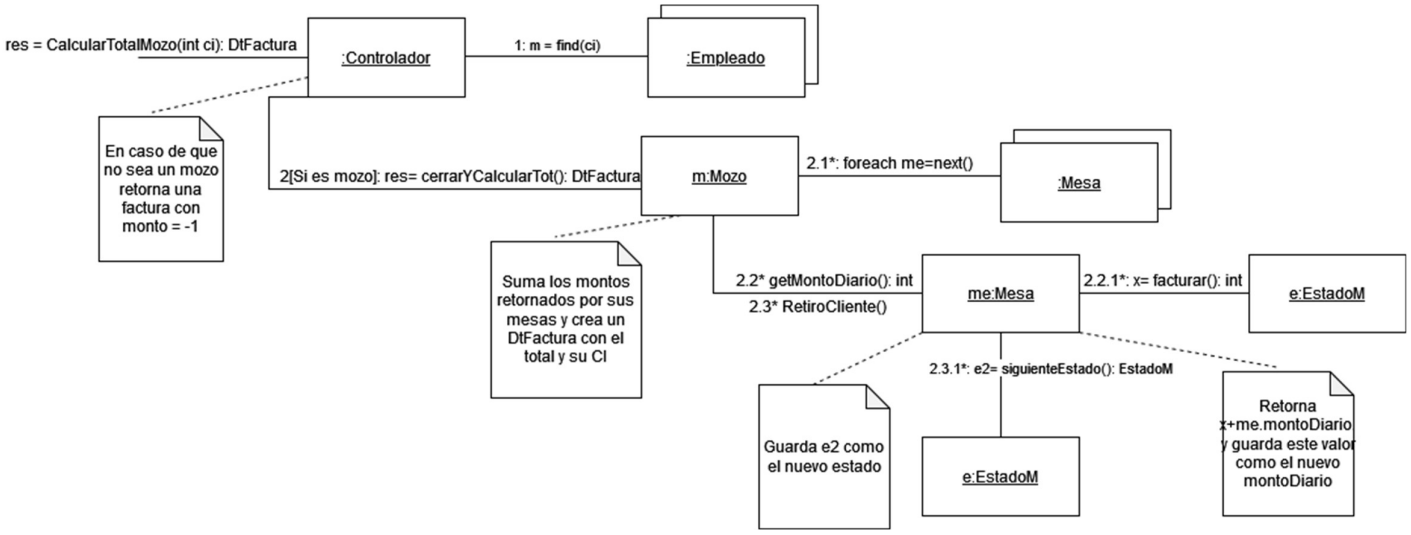


Figura 2

Considere los siguientes comportamientos de las operaciones:

- `agregarMesa`: Crea una nueva mesa con el id recibido como parámetro que queda asignada al mozo cuyo id es recibido como parámetro. Si no hay un mozo con id `idMo` o el mozo ya tiene una mesa con id `idMe` retorna `false` y no hace nada, en caso contrario retorna `true`.
- `nuevaMesa`: En caso de que el mozo no tenga una mesa con id `idM` crea una nueva mesa, la asocia al mozo y retorna `true`, de lo contrario retorna `false`.
- `facturar`: Retorna el monto en caso de ser `ConClientes` o 0 en caso de ser `Vacia`.
- `siguienteEstado`: Crea y retorna el estado contrario.

Observaciones:

- Puede utilizar colecciones genéricas (realizaciones de IDictionary e ICollection) o paramétricas (contenedores STL).
- No incluir directivas al precompilador (#include, etc).