Programación 4

PARCIAL FINAL EDICIÓN 2016

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 parcial junto al parcial
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el parcial

Problema 1 (30 puntos)

Dado el éxito a nivel mundial de los servicios de transporte privado de pasajeros utilizando capacidades ya instaladas (es decir, vehículos de particulares), un grupo de egresados de Fing ha decidido estudiar la viabilidad de un emprendimiento de estas características y le ha solicitado a usted realizar un análisis primario con el objetivo de desarrollar una aplicación móvil llamada "Yuber".

La aplicación será utilizada tanto por los usuarios choferes como pasajeros. Utilizando la misma, los pasajeros podrán realizar solicitudes de viaje. Dichas solicitudes podrán ser respondidas por los choferes que se encuentren próximos a la ubicación del pasajero que realiza la solicitud.

De los usuarios importa conocer su email, que debe ser único, su contraseña y su fecha de registro. En caso de tratarse de un usuario chofer, importará conocer también su nombre, edad y fecha de vencimiento de su licencia de conducir.

Para ofrecer el servicio, los choferes deben disponer de un vehículo, del que importa conocer el año, la matrícula (que lo identifica), la marca, el modelo y la capacidad de pasajeros. Dado que "Yuber "apunta a la excelencia en la calidad de su servicio, no se aceptan vehículos de años anteriores al 2011.

Cuando un pasajero realiza una solicitud de viaje, se registra si existen vehículos disponibles cercanos a su ubicación. En caso de que exista disponibilidad de vehículos, importa registrar además el tiempo de demora del chofer que responde a la solicitud. Debido a que la demanda de vehículos puede variar sustancialmente durante el día y durante la semana, se desea incorporar una tarifa dinámica, la cual, mediante un algoritmo complejo y propietario (de terceros), permitirá calcular un factor (que afectará el monto total del viaje) para dicha solicitud en función de la oferta y demanda actual. Solo en caso de que la solicitud sea confirmada por el pasajero, la misma tendrá asociada un viaje. Del viaje importa conocer su origen y destino (latitud y longitud), la duración y el monto total del mismo.

Por último, los pasajeros contarán con la posibilidad de utilizar distintos medios de pago para abonar el viaje. Por el momento se acepta únicamente una tarjeta internacional como medio de pago, pero se planea incorporar otros próximamente. De las tarjetas interesa conocer su número (identificador), código de seguridad y tipo (débito o crédito). No se podrá asociar un mismo medio de pago a distintos usuarios.

Además, se cuenta con el siguiente caso de uso:

Nombre	Solicitar Vehículo
Actores	Cliente
Descripción	Este caso de uso comienza cuando el cliente solicita un vehículo mediante el botón correspondiente en la aplicación. Esto genera una respuesta del sistema al cliente, la cual puede ser: **a) Sin Disponibilidad**: significa que actualmente no existen vehículos disponibles cercanos a la ubicación del cliente, por lo que el caso de uso termina con la cancelación de la solicitud. **b) Tarifa Normal**: significa que existen vehículos cercanos a la ubicación del cliente y se aplicará la tarifa normal, por lo que el cliente debe primero confirmar su solicitud, a lo cual el sistema responderá con el nombre del chofer, la matrícula del vehículo que se dirige a buscar al cliente y la cantidad de minutos que demorará. Posteriormente, en forma opcional puede ingresar la dirección de destino. **c) Tarifa Dinámica**: significa que existen muy pocos vehículos disponibles, por lo que se le aplicará una tarifa especial. Para conocer el factor de dicha tarifa, el cliente debe consultarlo al sistema mediante el botón correspondiente, a lo cual el sistema responderá con un número real que será multiplicado por el monto del viaje. Dado que este factor puede ser considerablemente alto, el cliente debe aceptar pagar ese factor o bien cancelar la solicitud. Si decide aceptar, también recibirá los datos referentes al nombre del chofer, matrícula del vehículo y minutos de demora, y tendrá la opción de ingresar el destino.

Se pide:

i. Modelo de Dominio de toda la realidad planteada (incluyendo información proporcionada en el Caso de Uso), con restricciones en lenguaje natural.

ii. Diagrama de Secuencia del Sistema para el Caso de Uso, incluyendo memoria del Sistema y uso de Datatypes en caso de ser necesarios.

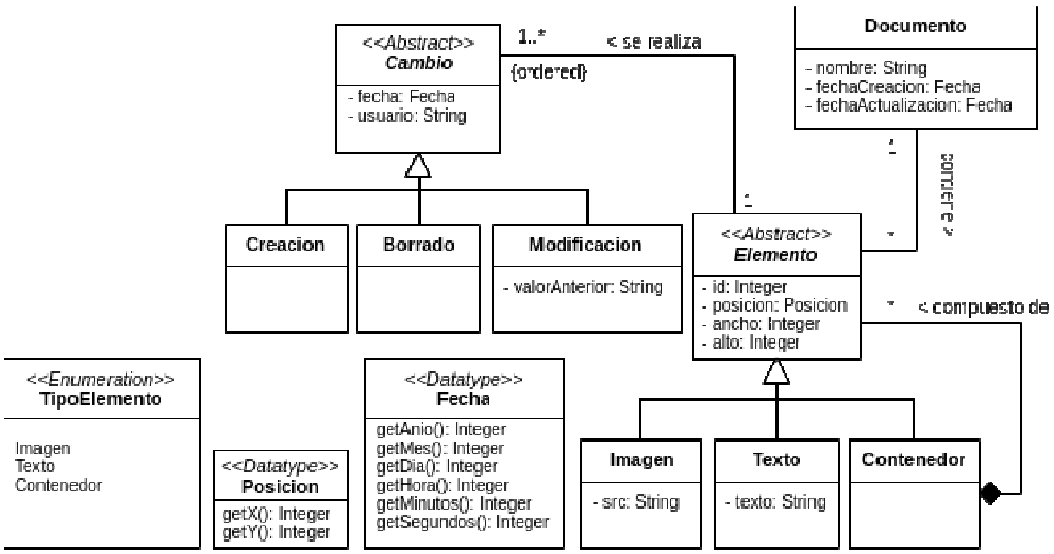
Problema 2 (35 puntos)

Parte a)

Realizar el Diagrama de Clases de Diseño completo del patrón de diseño Template Method.

Parte b)

Se desea implementar un editor de documentos sencillo. Cada documento estará compuesto de imágenes, texto, o contenedores, los cuales serán ubicados en el documento a través de coordenadas. Los cambios realizados en el documento serán registrados en el sistema. El modelo de dominio correspondiente a la realidad planteada se presenta en la siguiente figura.



Considere los siguientes contratos, correspondientes a operaciones de sistema:

crearElemento (tipo: TipoElemento, usuario, nomDoc: String, X, Y, ancho, alto, id: Integer)	
Descripción	Crea y agrega un nuevo *Elemento* sin contenido al *Documento*
Parámetros	– tipo: tipo de *Elemento* a crear. – usuario: nombre de usuario que realizó el cambio. – nomDoc: nombre del *Documento.* – x: coordenada x del *Elemento* en el *Documento*. – y: coordenada y del *Elemento* en el *Documento*. – ancho: ancho del *Elemento*. – alto: altura del *Elemento*. – id: identificador del *Elemento*
Pre-condiciones	– No existe un *Elemento* con identificador *id.* – Existe un *Documento* con nombre *nomDoc.*
Post-condiciones	– El Sistema genera una nueva instancia de *Elemento* con los datos indicados en los parámetros. Si el *Elemento* es una *Imagen, src* se inicializa como una cadena vacía. De la misma forma, si el *Elemento* es un *Texto,* el atributo *texto* se inicializa con una cadena vacía. – El Sistema crea una nueva instancia de *Creacion,* con atributo *fecha* igual a la fecha actual, y usuario igual al provisto como parámetro de la operación. – El Sistema genera un link entre el *Elemento* y la instancia nueva de *Creacion*. – El Sistema genera un link entre el *Elemento* y el *Documento.* – El Sistema actualiza el valor de *fechaActualizacion* en el *Documento* con la fecha actual del sistema.

eliminarElemento(id: Integer, usuario, nomDoc: String)	
Descripción	Elimina un *Elemento* del *Documento.*
Parámetros	– id: identificador del *Elemento* a eliminar. – usuario: nombre del usuario que realizó la modificación. – nomDoc: nombre del *Documento* a modificar.
Pre-condiciones	– Existe un *Elemento* con identificador *id.* – Existe un *Documento* con nombre *nomDoc.*
Post-condiciones	– El Sistema genera una instancia nueva de *Borrado* con fecha igual a la actual, y usuario igual al pasado por parámetro – El Sistema genera un link entre la nueva instancia de *Borrado* y el *Elemento* con identificador *id.* – Si el *Elemento* es un *Contenedor,* lo expresado en los dos puntos anteriores se repite para cada *Elemento* que lo componga. – El Sistema actualiza el valor de *fechaActualizacion* en el *Documento* con la fecha actual del sistema.

Se pide:

i. Realizar el Diagrama de Comunicación (incluyendo visibilidades) para cada una de las operaciones previamente especificadas.

ii. Realizar el Diagrama de Clases de Diseño correspondiente.

Problema 3 (35 puntos)

Su empresa ha decidido involucrarse en el área de reproducción de música por streaming. Para esto, le han encargado la implementación en C++ de la siguiente porción del sistema representado en el diagrama de clases que se adjunta (ver siguiente página).

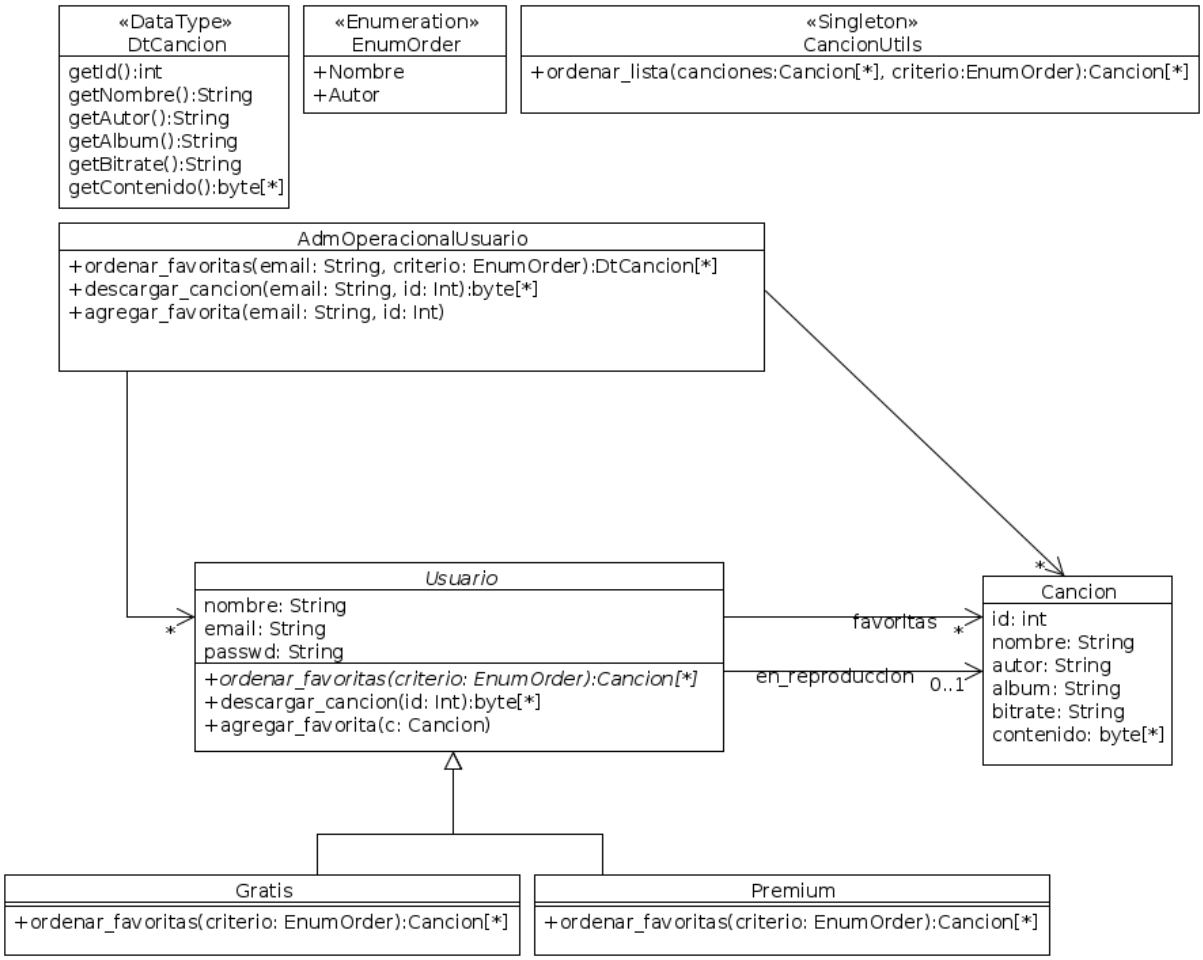
De esta manera, **AdmOperacionalUsuario** se encargará de una porción de las operaciones del sistema que involucran al usuario y sus canciones. Este contendrá la colección de todos los usuarios así como de todas las canciones existentes.

La siguiente es una descripción de cada una de sus operaciones**.**

descargar_cancion: Dado un identificador de usuario y uno de canción, retorna el contenido de dicha canción. Si la canción no pertenece a las favoritas del usuario, lanzará la excepción **invalid_argument**. Asumir la existencia del usuario dado su identificador.

agregar_favorita: Dados un identificador de usuario y de canción, agrega esta última a la colección de favoritas del usuario. Puede asumir la existencia en el sistema de la canción y del usuario.

ordenar_favoritas: Dado un identificador de usuario y un criterio de ordenamiento, retorna la colección de canciones favoritas del usuario ordenada por este criterio. Asumir la existencia del usuario dado su identificador. El retorno debe ser una colección de **DtCancion**. Esta operación solo podrá ser ejecutada por usuarios **Premium**, mientras que para usuarios **Gratis** se lanzará la excepción **invalid_argument**.



Considerar:

- La clase **CancionUtils** es un Singleton que provee una función tal que dadas una colección de canciones y un criterio de ordenación, retorna una colección que es copia de la ingresada por parámetro y está ordenada según el criterio especificado. Considerar la operación como implementada.
- El tipo **byte** puede ser representado en C++ por el tipo básico **char**.
- Cada **Usuario** es identificado por su **email** y cada **Canción** por su **id** entero.
- Puede suponer la existencia de la interface **ICollectible** e implementaciones de **IDictionary** (clase **OrdererDictionary**), **Ikey**, de la clase **List** e **IIterator** según sea necesario.
- Es posible utilizar las clases **set<T>**, **list<T>** y **map<K,T>** de la STL.
- Las implementaciones deben incluir constructores y destructores.
- Implementar los getters solamente para datatypes.
- No implementar setters.
- No incluir directivas al precompilador.

Se pide:

i. Implementar todos los **.h** y **.cpp** de las clases que aparecen en el diagrama.
ii. Implementar el Singleton **CancionUtil**, sin dar método a la operación **ordenar_lista**.
iii. Implementar el datatype **DtCancion**.
iv. Definir el enumerado **EnumOrder**.