Programación 4

EXAMEN JULIO 2017

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

Problema 1 (35 puntos)

Se desea implementar una red social para los alumnos de la Fing, FingSocial, donde puedan compartir sus experiencias sobre las materias de la facultad. Para utilizar el sistema, los alumnos deben registrarse como usuarios del mismo.

De cada usuario registrado interesa conocer su email que lo identifica, su nombre y su fecha de nacimiento. Además, se cuenta con información de las materias dictadas en la Facultad, de las cuales se tiene el nombre, el semestre en que se dictan (par o impar) y un hashtag (por ejemplo #programacion4) único para cada materia.

Los usuarios pueden realizar dos tipos de actividades dentro de la red social, publicaciones o comentarios, en los cuales pueden mencionar a las materias, utilizando hashtags. Tanto para las publicaciones como para los comentarios, interesa registrar un identificador numérico, la fecha en que se realizaron, el texto ingresado y qué materias se mencionan. De los comentarios interesa además conocer a qué publicación o comentario responden, ya que pueden haber comentarios anidados. Por otra parte, las publicaciones pueden hacer referencia a otras publicaciones. Además éstas pueden incluir una foto de la cual se precisa saber su url.

Se desea además que los usuarios puedan indicar cómo se sintieron acerca de una publicación o un comentario hecho por otro alumno, e interesa registrar si fue un sentimiento negativo o positivo, y la fecha en que se realizó.

Por último, los usuarios pueden registrar información sobre las materias que cursaron. Dado que un alumno puede haber cursado una materia más de una vez, se pueden indicar todas las veces en que se cursó una materia, especificando si se está cursando actualmente, o en caso contrario, el año en que se cursó. Opcionalmente se pueden indicar las pruebas que se realizaron en cada curso, tanto previos como actuales, indicando el tipo (laboratorio, parcial o examen) y la nota obtenida.

Caso de Uso:	EditarComentario
Actor:	Usuario
Descripción:	El caso de uso comienza cuando el usuario indica que quiere editar un comentario previamente hecho. El sistema entonces lista los comentarios hechos por el usuario, y éste elige uno. A continuación, el sistema lista las materias existentes y mientras desee, el usuario puede elegirlas para incluir los hashtags en el comentario. Luego el usuario ingresa el texto modificado y el sistema actualiza el comentario, modificando también la fecha almacenada.

Se pide:

i) Modelo de Dominio de la realidad anterior con restricciones en lenguaje natural. Se deben especificar los tipos de los atributos, así como también los Datatypes y enumerados utilizados.

ii) Diagrama de Secuencia del Sistema (DSS) del Caso de Uso, incluyendo el uso de Datatypes y de manejo de memoria del Sistema, en caso de ser necesario.

Problema 2 (35 puntos)

Parte A:

a) Describir dos criterios GRASP vistos en el curso.

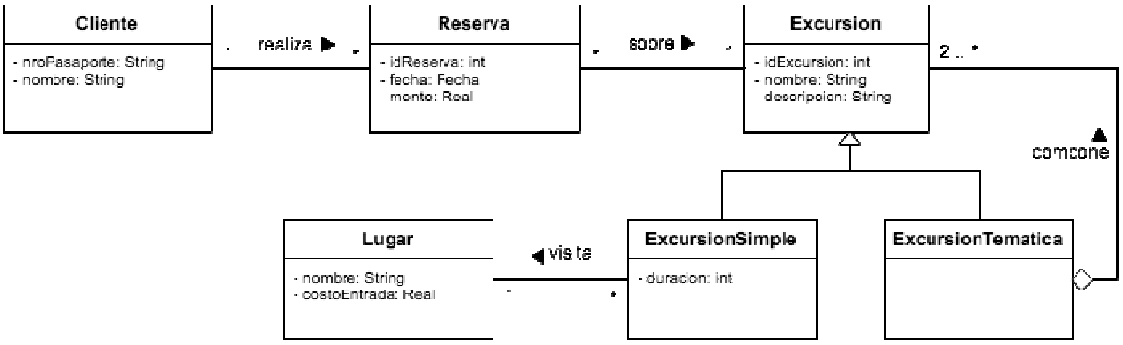
b) De los tipos de visibilidad aplicables en diagramas de comunicación de UML indicar cuáles son temporales.

Parte B:

Una compañía de turismo se dedica a la venta de excursiones a diferentes lugares de la ciudad. Las excursiones se clasifican en excursiones simples, en las que se visita un solo lugar, y en excursiones temáticas, compuestas por varias excursiones.

Por ejemplo, se ofrece una excursión simple al "Museo de Bellas Artes" y otra al "Museo de Historia Nacional" y por otro lado una excursión temática de arte donde se visitan ambos museos.

Cada excursión tiene un identificador y un conjunto de reservas realizadas por los clientes. De los lugares se conoce el nombre que lo identifica y el costo de la entrada. Finalmente los clientes son identificados por su número de pasaporte.



Se pide:

i) Realice el diagrama de comunicación, incluyendo visibilidades y siguiendo los criterios GRASP, de la siguiente operación:

`void realizarReserva (nroPasaporte:String, idExcursion: int)`	
Descripción	Permite a un cliente realizar una reserva de una excursión.
Parámetros	- *nroPasaporte*: Identifica al cliente - *idExcursión*: Identifica a la excursión
Precondiciones	- Existe en el Sistema una única instancia de Cliente con número de pasaporte *nroPasaporte*. - Existe en el Sistema una única instancia de Excursión con identificador *idExcursión*.
Postcondiciones	- Se crea una nueva instancia de Reserva con id autogenerado, fecha y hora del sistema y monto calculado de la siguiente forma: • si la Excursión es simple el monto está dado por el costo de la entrada del lugar visitado. • si la Excursión es temática el monto está dado por la suma de los costos de cada excursión que la compone. - Se crea link entre la instancia de Cliente y la instancia de Reserva. - Se crea link entre la instancia de Reserva y la instancia de Excursión.

ii) Realizar un Diagrama de Clases de Diseño (DCD) con el diseño propuesto para soportar el modelo de dominio presentado y las operaciones diseñadas en la parte anterior. En caso de utilizar un patrón de diseño, indicar su nombre e incluir las clases participantes y sus roles.

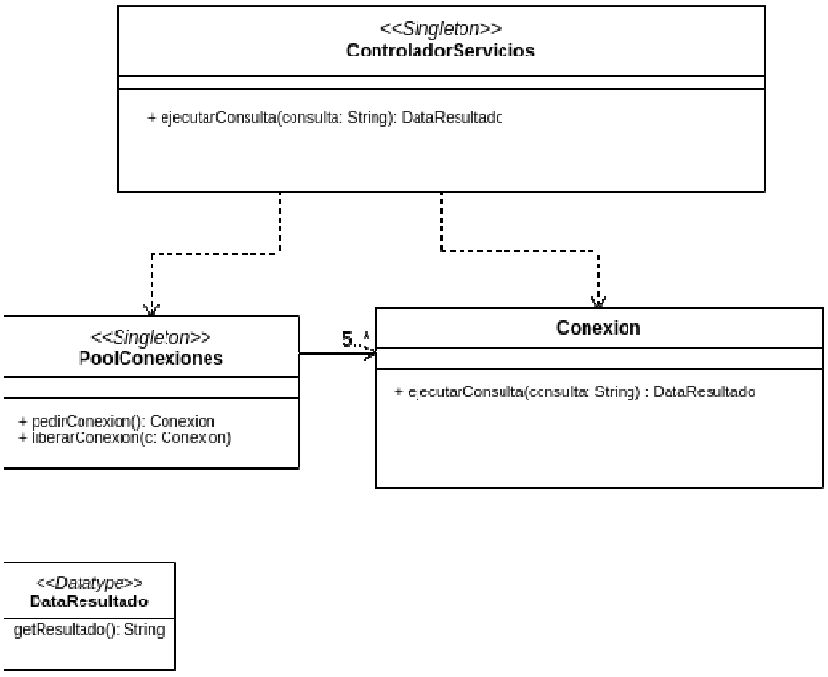
Aclaraciones:
- Incluir en el DCD únicamente los setters y getters que utilice en la parte (i).
- Incluir en el DCD únicamente los constructores y destructores que utilice en la parte (i).

Problema 3 (30 puntos)

La instanciación de ciertos tipos de objetos (por ejemplo, los que manejan las conexiones a la base de datos de un servidor web) puede representar un cuello de botella en algunas aplicaciones. El patrón de diseño *Object pool* contribuye a reducir el impacto negativo en el desempeño de un sistema debido al costo implícito en la creación de estos tipos de instancias. El mismo consiste en:

- Un repositorio (pool) de objetos reusables.
- Un cliente, que pide un reusable al pool, y lo devuelve cuando no lo necesita más.
- Los reusables, que son instancias cuyo costo de creación (en tiempo o utilización de CPU) es alto, y cuya frecuencia de creación y destrucción en la aplicación justifica mantener instancias siempre disponibles. El pool se encarga de la instanciación y destrucción de instancias de reusables.

Considere el diagrama de clases de la imagen:



La operación *ejecutarConsulta* de *ControladorServicios* recibe la consulta a ejecutar, y devuelve un *DataResultado* con el resultado de la misma. La consulta se realiza obteniendo una instancia de *Conexion* y llamando a su operación *ejecutarConsulta*.

Tener en cuenta que por cada llamada a *ControladorServicios::ejecutarConsulta* debe haber una instancia de *Conexion* disponible, que de no existir debe ser creada. El control de la concurrencia de las invocaciones a la operación del controlador es resuelta por clases externas al diagrama dado.

Considerar también que la búsqueda, instanciación, o liberación de instancias de *Conexion* en *PoolConexiones* debe realizarse de la forma más rápida posible, por razones de rendimiento del sistema.

Considerar:

- Puede suponer la existencia de la interface *ICollectible* e implementaciones de *ICollection* (clase *List*), *Ikey* (clase *OrderedKey*), *IDictionary* (clase *OrderedDictionary*) *e IIterator* según sea necesario.
- Es posible utilizar las clases *set<T>*, *vector<T>* o *map<K,V>* de la *STL*.
- Las implementaciones **deben** incluir constructores y destructores.
- Implementar los getters **solamente** para datatypes.
- **No** implementar setters y getters de las clases del sistema.
- **No** incluir directivas al precompilador.

Se pide:

i) Implementar .h y .cpp del data type *DataResultado*.
ii) Implementar .h y .cpp de la clase *ControladorServicios*.
iii) Implementar .h y .cpp de la clase *PoolConexiones*.
iv) Implementar .h y .cpp de la clase *Conexion*. La operación *Conexion::ejecutarConsulta* no debe hacer nada más que devolver una instancia de *DataResultado*. No se espera nada más en su método.