Programación 4

EXAMEN FEBRERO 2019

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 (30 puntos)

Parte a:

Un Diagrama de Secuencia del Sistema (DSS) se construye a partir de un Caso de Uso (CU) Expandido, tomando algunas de las interacciones del CU y convirtiéndolas en Operaciones del Sistema.

Se pide:

Indicar qué criterio se utiliza para decidir si una interacción de un CU se convierte en una Operación del Sistema en el DSS.

Parte b:

Un metamodelo de UML es un modelo que describe otro modelo, utilizando los mismos elementos UML. Por ejemplo, se puede realizar un metamodelo de un diagrama de clases mediante otro diagrama de clases. El metamodelo describe las reglas sobre cómo se construye un diagrama de clases.

Se desea realizar un modelo de dominio que represente a los diagramas de clases de UML. O lo que es lo mismo: realizar el metamodelo de los diagramas de clases, utilizando para ello un diagrama de clases. De todos los elementos que puede contener un diagrama de clases, solamente se desea modelar los siguientes: clases (abstractas y concretas), atributos (incluyendo su nombre, visibilidad, tipo, multiplicidad y si es de clase o de instancia), generalizaciones, asociaciones entre dos clases (incluyendo su nombre, si es una composición o agregación, los dos roles y sus multiplicidades), operaciones (incluyendo su nombre, visibilidad y si es abstracta) y parámetros de operaciones (incluyendo su nombre y tipo) y tipo de retorno de la operación.

Se pide:

Realizar el metamodelo de los diagramas de clases, restringiéndose únicamente a los elementos mencionados. Incluir las restricciones correspondientes en lenguaje natural.

Problema 2 (35 puntos)

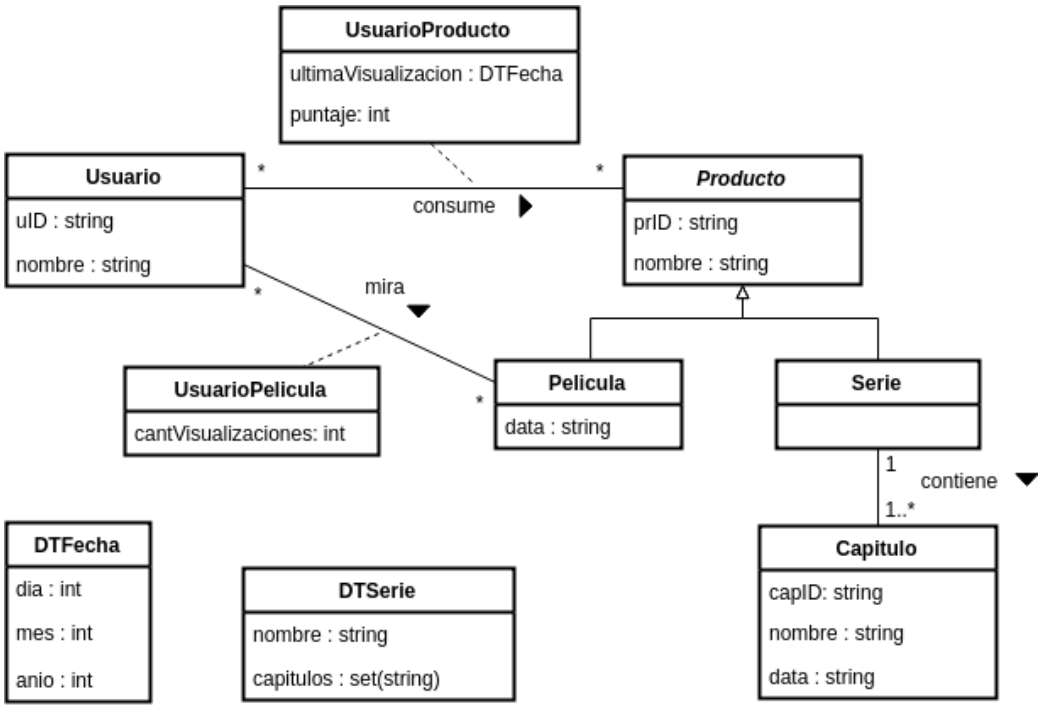
Parte a:

i) Nombre dos criterios GRASP y explique brevemente qué significa cada uno.

ii) Explique el problema tipo que resuelve el patrón Observer e ilustre gráficamente su estructura general.

Parte b:

La empresa ***Netfling*** le ha encomendado a usted el desarrollo de una aplicación que tiene usuarios que se registran a la misma y consumen productos. Como se puede apreciar en el modelo a continuación, un producto es o bien una serie o una película. Para cualquier producto interesa saber qué usuario lo ha visto, y en ese caso se mantiene la fecha de la última visualización y el puntaje de satisfacción que ha dado sobre el mismo. Por otra parte, interesa saber para cada usuario cuántas veces ha visualizado una película.



Se pide:

i)

Realizar el Diagrama de Comunicación correspondiente a la siguiente operación del sistema, **incluyendo visibilidades**:

`obtenerInfoSerie(prID: string): DTSerie`	
Descripción	La función retorna la información de una serie.
Parámetros	***`prID`***: El identificador en el sistema de la serie.
Precondiciones	Existe en el sistema una serie con identificador ***`prID`***.
Postcondiciones	Se retorna un ***`DTSerie`*** con el nombre de la serie y con los nombres de los capítulos que existen para la misma.

ii)

Realizar el Diagrama de Comunicación correspondiente a la siguiente operación del sistema, **incluyendo visibilidades**:

`verYPuntuarPelicula(uID: string; prID: string, p: int)`	
Descripción	Se registra una nueva visualización y asignación de puntaje a una película, por parte de un usuario.
Parámetros	***uID***: El identificador en el sistema del usuario. ***prID***: El identificador en el sistema de la película. ***p***: El puntaje que le asigna el usuario a la película.
Precondiciones	Existe en el sistema una película con identificador ***prID*** y un usuario con identificador ***uID***.
Postcondiciones	Si el usuario ya vió previamente la película: • Se aumenta en 1 la cantidad de veces que el usuario vió la película en la instancia de `UsuarioPelicula` correspondiente. • Se actualiza la `ultimaVisualizacion` del producto en la instancia de `UsuarioProducto` correspondiente, con la fecha del sistema, y se actualiza el puntaje con el valor `p`. De lo contrario: • Se crea una instancia de `UsuarioPelicula` con la cantidad en 1. • Se crea una instancia de `UsuarioProducto` con el puntaje `p`, y la `ultimaVisualizacion` con la fecha del sistema. • Se crean los enlaces correspondientes entre las nuevas instancias, el usuario y la película.

Nota:

Puede asumir que el controlador que defina tiene una operación **`obtenerFechaActual():DTFecha`**, que retorna la fecha actual del sistema.

iii)

Realizar el Diagrama de Clases de Diseño (DCD) que soporte las partes i) y ii).

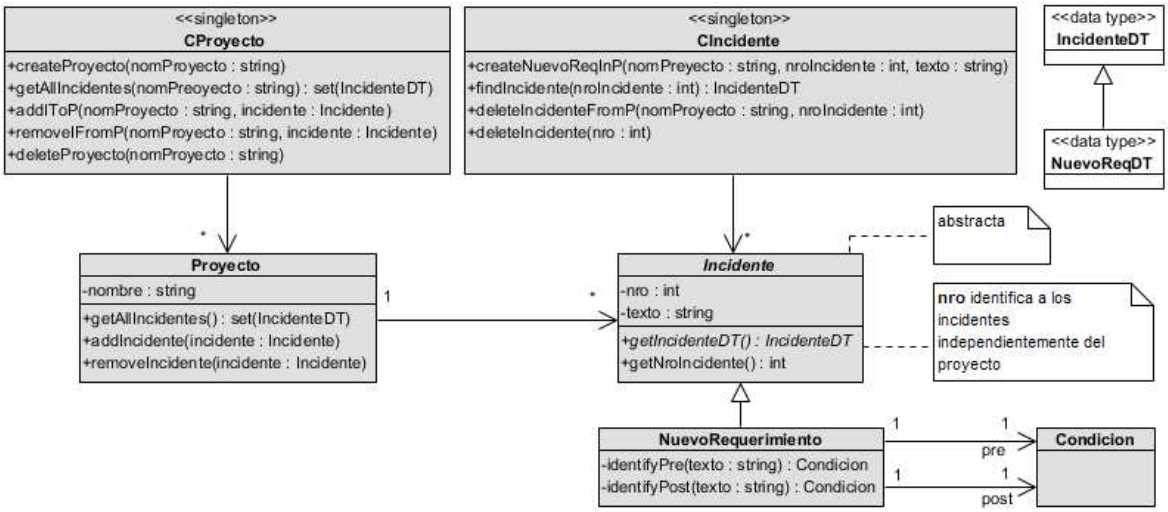
Notas:

- No incorporar setters ni getters al DCD.
- Incorporar en el DCD sólo los constructores o destructores que se utilicen en los diagramas de comunicación.
- No incorporar al DCD las operaciones de colecciones.

Problema 3 (35 puntos)

Un gestor de incidentes de software es un sistema que mantiene registro de incidentes reportados en proyectos de desarrollo de software. Un incidente puede ser un error, un nuevo requerimiento o una mejora, entre otros.

Considere la siguiente porción del diseño de un prototipo de un gestor de incidentes de software correspondiente al ciclo de vida de los objetos con información de los proyectos y sus incidentes, dada por el diagrama de clase de diseño (DCD) de abajo.



La responsabilidad de creación y destrucción de proyectos así como la gestión de la colección de proyectos está asignada a **`CProyecto`**. Esta asignación se lleva a cabo mediante las operaciones **`createProyecto`** y **`deleteProyecto`**. `CProyecto` se ocupa además del vínculo de cada proyecto con sus incidentes mediante las operaciones **`addIToP`**, que vincula un incidente con un proyecto, y **`removeIFromP`,** que desvincula al incidente del proyecto. Estas operaciones hacen uso de las operaciones **`addIncidente`** y **`removeIncidente`** de **`Proyecto`**, respectivamente, para cumplir su función.

A su vez, la responsabilidad de creación y destrucción de incidentes así como la gestión de la colección de incidentes está asignada a **`CIncidente`**. Esta asignación se cumple por medio de las operaciones **`createNuevoReqInP`** y **`deleteIncidenteFromP`** que reciben entre sus parámetros el nombre del proyecto al cual debe ser vinculado/desvinculado el incidente que se crea/destruye, respectivamente. `CIncidente` posee además la operación **`deleteIncidente`** que elimina el incidente (identificado por su número) de su colección y lo destruye.

Por último, con respecto a la asignación de responsabilidades, se decidió que al destruir un proyecto se destruyen también todas sus incidencias vinculadas. Esta decisión se quiere respetar sin agregar nuevas operaciones a las que ya están en el DCD dado.

A los efectos de este ejercicio, asuma que se cumplen las precondiciones de todas las operaciones (por ej. el nombre de un proyecto a crear no existe en la colección de proyectos). También asuma que la operación **`getIncidenteDT`** de la clase **`Incidente`** así como las operaciones **`identifyPre`** e **`identifyPost`** de la clase **`NuevoRequerimiento`** ya están implementadas. A partir del texto del incidente escrito en un lenguaje de especificación de requerimientos, `identifyPre` devuelve un nuevo objeto **`Condicion`** que contiene una representación ejecutable de la precondición. `identifyPost` es análoga.

Se pide implementar en C++:

i) `CProyecto`: .h y .cpp completos.
ii) `CIncidente`: .h completo y del .cpp sólo las operaciones `createNuevoReqInP`, `deleteIncidenteFromP` y `deleteIncidente` (**NOTAR** que no se pide `findIncidente`).
iii) `Proyecto`: .h y .cpp completos.
iv) `Incidente`: .h y .cpp completos.
v) `NuevoRequerimiento`: .h completo y del .cpp sólo el constructor no vacío y el destructor. El constructor no vacío recibe como parámetros el nro de incidente y el texto.

Observaciones:

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