Programación 4

EXAMEN FEBRERO 2014

Por favor, siga las siguientes indicaciones:

- Escriba con lápiz
- Escriba las hojas de un solo lado
- 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

Problema 1 (30 puntos)

a. Definir los conceptos *evento del sistema* y *operación del sistema* en el contexto de la etapa de análisis.

b.

La compañía *Muebles Modernos* se dedica al desarrollo de muebles armados en base a componentes estándar disponibles en el mercado. Existe un departamento que diseña los productos, que se identifican por su nombre; además se debe saber el año de diseño del producto y su diseñador. Cada producto tendrá un proyecto asociado (inicialmente puede no tener ninguno), del que se conoce su código (que lo identifica) y su proyectista. Un proyecto comprende varios componentes, identificados por un número y que pueden ser de tipo estructural o de cerramiento. Un proyecto es llevado a cabo por uno o varios constructores, que deben tener habilidad en la manipulación de los componentes que comprende el proyecto. Para cada constructor se sabe su grado de habilidad en cada componente, representado como un número entero no negativo en el rango 0..5; se consiera que un constructor tiene habilidad sobre un componente si su respectivo valor asociado es mayor o igual a 2. Tanto los diseñadores, como los proyectistas y constructores, son identificados por su número de documento de identidad.

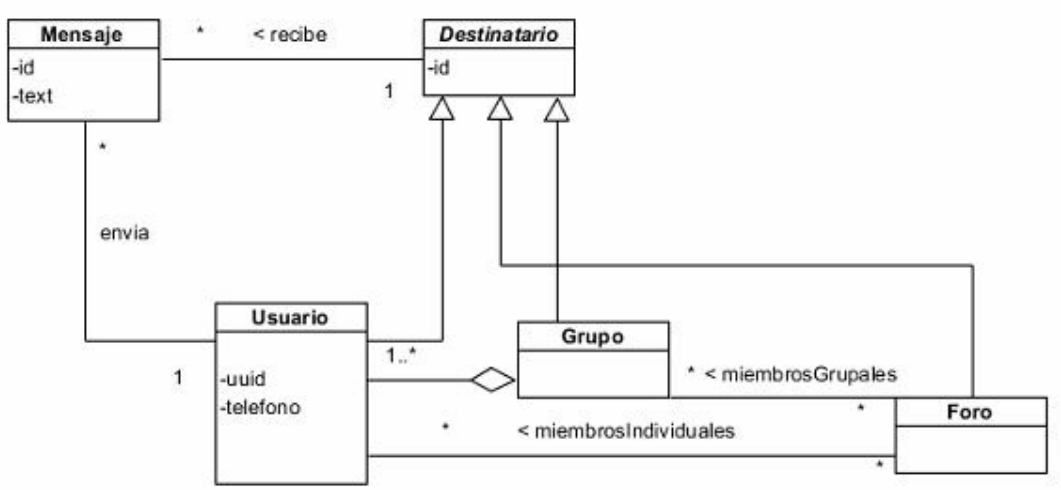
Se pide:

i. Realizar el Modelo de Dominio en notación UML.
ii. Escribir en lenguaje natural, todas las restricciones no estructurales que sean necesarias.

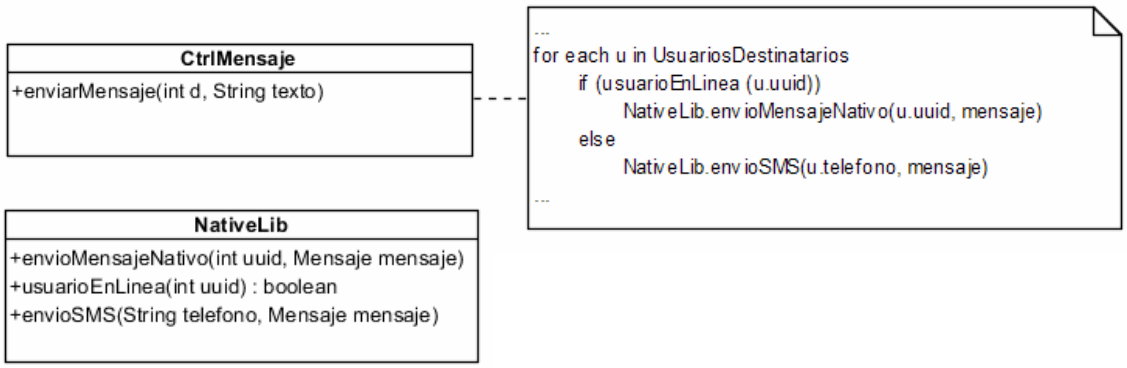
Problema 2 (35 puntos)

a. Enumere los criterios GRASP.

b. Un sistema de mensajería instantánea para dispositivos móviles se basa en el siguiente modelo de dominio.



Los usuarios pueden enviarse mensajes entre sí o formar parte de grupos o foros para recibir mensajes que envían otros usuarios. Considere que un mensaje enviado a un foro, debe llegar tanto a sus miembros grupales como a sus miembros individuales. Para ello se decide crear la clase `CtrlMensaje` que implementa *Singleton* y que debe incluir la operación `enviarMensaje`. Esta operación debe utilizar a otro *Singleton* denominado `NativeLib`, para acceder a funcionalidades nativas del dispositivo. En la nota de ejemplo a continuación, se ilustra el procedimiento para el envío de mensajes donde se verifica si el usuario está en línea para definir el tipo de mensaje (nativo o SMS) que se le enviará. Tenga en cuenta que la lista de usuarios destinatarios dependerá del tipo de destinatario.



Se pide:

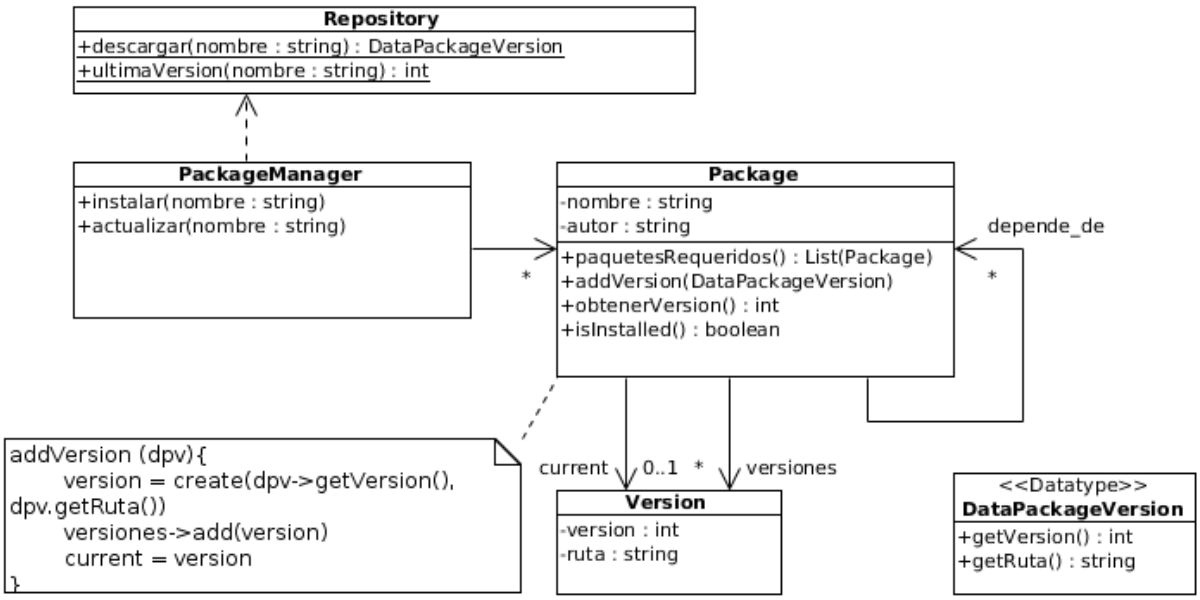
i. Realice el diagrama de comunicación de la operación `enviarMensaje` realizando un filtrado de modo que un usuario al que le corresponda recibir el mensaje, lo reciba una única vez, aún estando asociado a más de un grupo o foro que le corresponda recibir dicho mensaje. Asuma las operaciones de `NativeLib` como primitivas.

ii. Se desea contar con una forma alternativa de envío de mensajes a usuarios que forman parte de grupos y foros. Para ello se concibe un mecanismo de suscripción independiente a las asociaciones de grupo y foro dadas por el modelo de dominio. Es decir, un mensaje enviado a un grupo o foro deberá llegar solamente a los usuarios suscritos a estos. Utilizando los patrones de diseño presentados en el curso, realice un DCD que provea un mecanismo genérico para la suscripción y envío de mensajes de todos los usuarios suscritos a grupos o foros. Indique qué patrón(es) utilizó, qué roles cumplen las clases involucradas y proporcione el pseudocódigo de las operaciones relevantes.

Problema 3 (35 puntos)

Un *Gestor de Paquetes* es una herramienta que permite instalar, actualizar y gestionar paquetes de software en un sistema operativo de forma consistente. Típicamente mantiene una base de datos con información sobre las versiones de los paquetes y sus dependencias, para asegurar que a la hora de instalar un nuevo paquete, el sistema operativo cumple con todas sus dependencias.

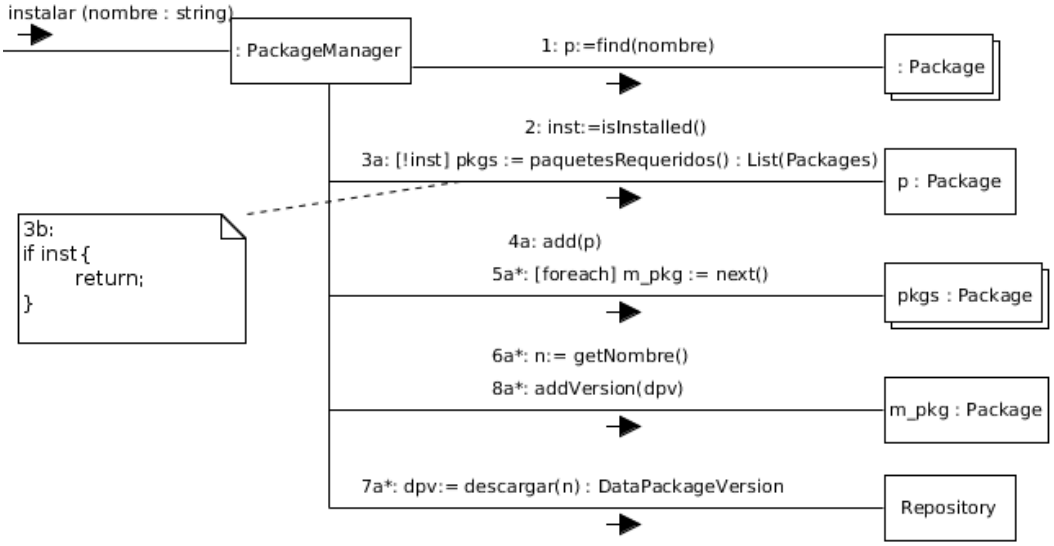
Le fue encomendada la implementación de un gestor con estas características, que permite instalar y actualizar paquetes de software, así como facilitar el manejo de las dependencias entre los mismos. Como resultado de la etapa de diseño se elaboró un Diagrama de Clases de Diseño parcial que se presenta a continuación:

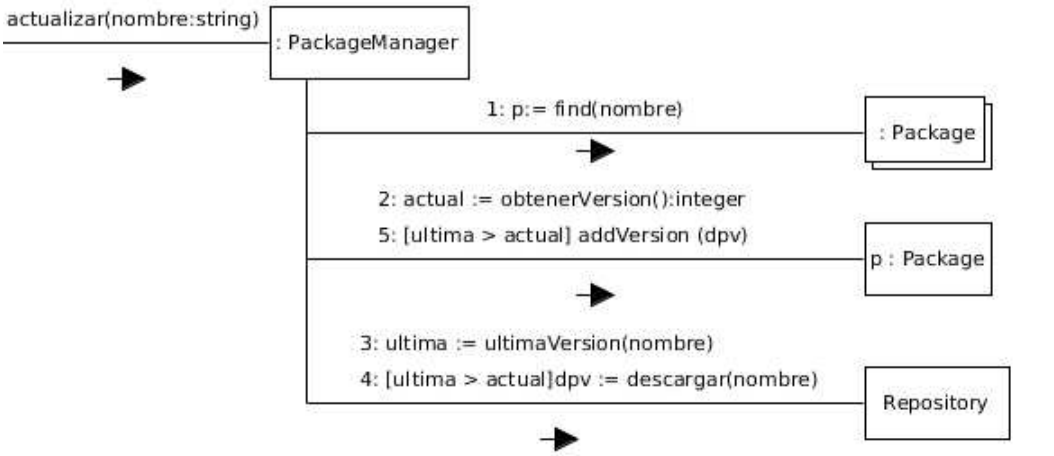


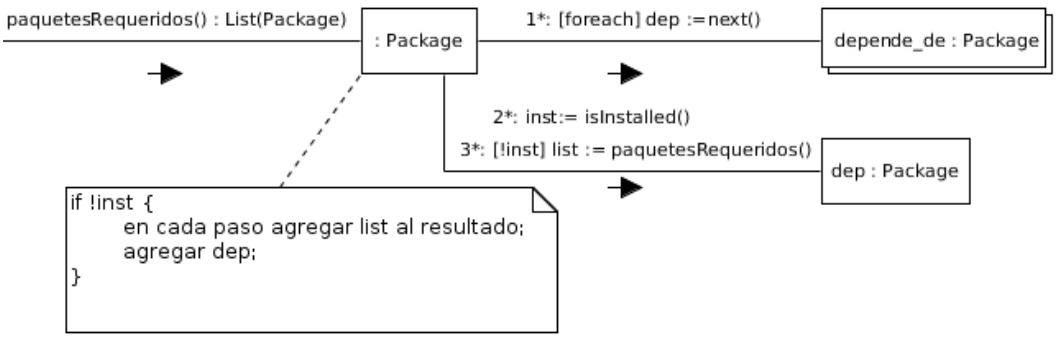
La clase `PackageManager` tiene un diccionario de paquetes que se identifican por su nombre e implementa las siguientes operaciones:

- `instalar`, que permite instalar un paquete a partir de su nombre.
- `actualizar`, que instala una versión más reciente del paquete en caso de que exista.

A su vez, la clase `Package` mantiene la información relacionada con un paquete, entre la que se encuentra su nombre, autor, sus dependencias y una colección con las distintas versiones instaladas del mismo. La operación `obtenerVersion` devuelve el atributo `version` de la instancia `current`, mientras que la operación `isInstalled` devuelve true en caso de que la colección de versiones no sea vacía. Su equipo recibió además un Diagrama de Comunicación que detalla algunas de las operaciones que debe implementar:



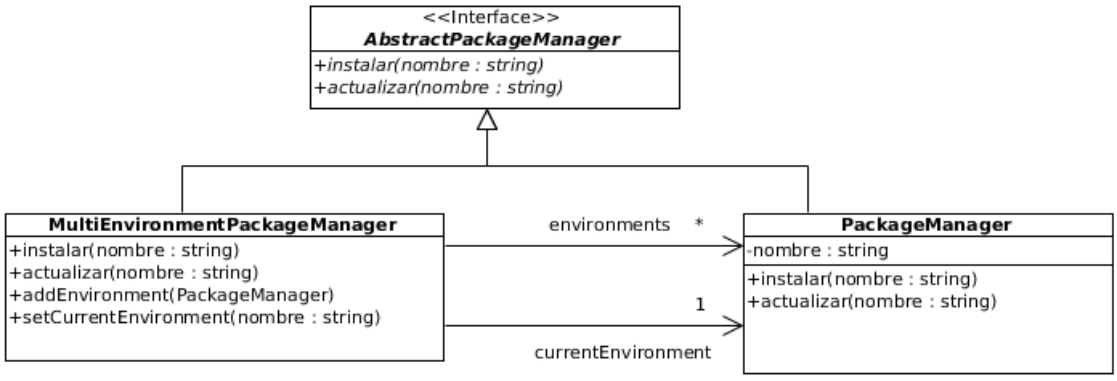




Se pide:

a. Implementar en C++ las clases `PackageManager` y `Package`, incluyendo constructores y destructores. No implementar las operaciones `get` y `set`.

Luego de las primeras pruebas de la herramienta, surgió la necesidad de adaptarla para permitir gestionar ambientes de desarrollo con distintos paquetes instalados. Para ello se elaboró el siguiente diagrama:



`MultiEnvironmentPackageManager` permite mantener distintos ambientes aislados, e instalar diferentes paquetes en cada uno. Para ello mantiene un diccionario de `PackageManager` que se identifican por su nombre. Delega la instalación y actualización de paquetes al `PackageManager` actual (`currentEnvironment`).

Se pide:

b. Implementar en C++ las clases `AbstractPackageManager`, y `MultiEnvironmentPackageManager`, incluyendo constructores y destructores. No implementar las operaciones `get` y `set`.

Observaciones:

- No incluir directivas al precompilador.
- Puede suponer la existencia de implementaciones de `IDictionary`, `ICollection`, `IIterator` y `KeyString` según sea necesario.
- Es posible utilizar las clases `set<T>` y `map<K,V>` de la STL.
- Puede asumir que no existen dependencias cíclicas entre los paquetes, es decir que siempre va a ser posible satisfacer las dependencias de los mismos.