# Programación 4

EXAMEN DICIEMBRE 2012

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** (35 puntos)

a. Indique los componentes necesarios para la especificación de un Caso de Uso Expandido.

b. Se desea implementar un software de gestión para una empresa de salas de entretenimiento. La empresa cuenta con múltiples locales que se identifican por un nombre y de los cuales se registra su dirección y número de teléfono. Cada local cuenta con múltiples salas de cine y teatro. Se conoce la capacidad de cada una de las salas, si poseen aire acondicionado o no y, para las salas de teatro, la potencia del sonido medida en KW. La asignación de las obras y películas exhibidas en cada sala (teatro y cine, respectivamente) se realiza en base semanal. Cada semana es conocida por un número del 1 al 52. Una obra o película en una semana se asigna a una única sala pudiéndose exhibirse en ella hasta en 3 funciones por semana. La funcion ofrece la informacion del dia (refiriéndose al nombre del día y no a una fecha, por ejemplo martes) y de la hora determinada en que se exhibe una obra o película. Para cada una de las funciones interesa registrar la cantidad de entradas vendidas. De cada película y obra se conoce su titulo y director y, en particular para las películas, interesa saber su origen.

   **Se pide**:

   i. Modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML.
   ii. Expresar todas las restricciones del modelo en **lenguaje natural**.

c. Se quiere optimizar el uso de las salas por lo que se desea asignar distintas salas a las funciones semanales asignadas a una obra o película y no en una única sala asignada semanalmente como se restringe en (b). Además, debido a las numerosas quejas por colas innecesarias, se desea numerar todas las entradas vendidas por lo que el sistema deberá permitir registrar y consultar la disponibilidad de cada asiento para cada función.

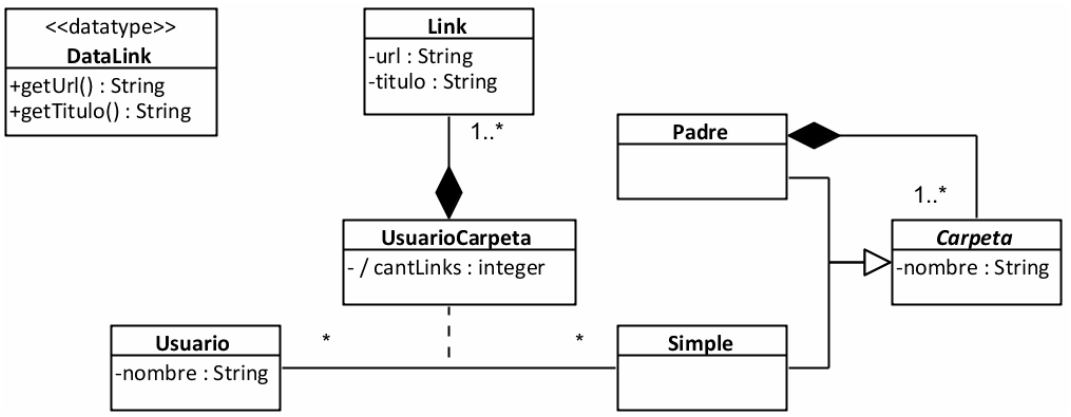
   **Se pide**:

   iii. Modelar esta nueva realidad mediante un Diagrama de Modelo de Dominio UML explicitando las modificaciones necesarias al modelo de la parte (b).
   iv. Expresar todas las restricciones del modelo en lenguaje natural.

## Problema 2 (35 puntos)

Se desea construir un sistema para un sitio web de almacenamiento de enlaces donde los usuarios pueden compartir *Links* interesantes a páginas web y almacenarlos en diferentes carpetas que son visibles desde todo el sitio. Las carpetas son compartidas y los usuarios pueden subir links en las carpetas independientemente de que otro usuario haya subido algún link en ella. Las carpetas forman una estructura arborescente de manera similar a un sistema de archivos. Hay dos tipos de carpetas: las *Carpetas Simples* que son las que contienen los Links y las carpetas *Padre* que agregan a un conjunto de carpetas.

De cada *Link* se conoce la URL a la página web (que lo identifica dentro de la carpeta) y su título. De los usuarios se conoce el nombre que los identifica. Es de interés saber para cada usuario cuantos links en total subió a cada carpeta. Los links pueden estar solamente en Carpetas Simples. De una carpeta se conoce su nombre que la identifica.
De acuerdo a los requerimientos el equipo de analistas realizó un modelo de dominio que se deberá respetar durante el diseño.




La intención es que el equipo que usted integra diseñe la siguiente operación del sistema.

| **`getLinks(usr: String, N: integer, carp: String):Set(DataLink)`** | |
|---|---|
| Descripción | Obtiene los datos de los links subidos en Carpetas Simples contenidas en la carpeta de nombre `carp` por el usuario de nombre `usr`.<br><br>Sólo se consideran los links subidos en Carpetas Simples donde el usuario haya subido al menos `N` links.<br><br>Si la carpeta `carp` es Simple sólo se busca en esa carpeta. De lo contrario se busca en las **todas las** carpetas hijas de `carp`. |
| Precondiciones | Existe una instancia de Usuario con nombre `usr`<br>Existe una instancia de Carpeta (Simple o Padre) de nombre `carp` |
| Postcondiciones | Devuelve un Set con data values de DataLink que correspondan a los Links que están asociados a una instancia de UsuarioCarpeta `uc` que cumple que<br>• `uc` está asociada al usuario de nombre `usr`<br>• `uc.cantLinks` ≥ `N`<br>• `uc` está asociada con una instancia de carpeta Simple de nombre `carp` o `uc` está asociada con una instancia de carpeta Simple que es hija de una carpeta de nombre `carp`. |

**Se pide**

1. Realizar el Diagrama de Comunicación de getLinks().
    - Indicar visibilidades en los diagramas
    - Indique claramente el resultado de cada operación utilizando notas
2. Realizar el Diagrama de Clases de Diseño de la solución

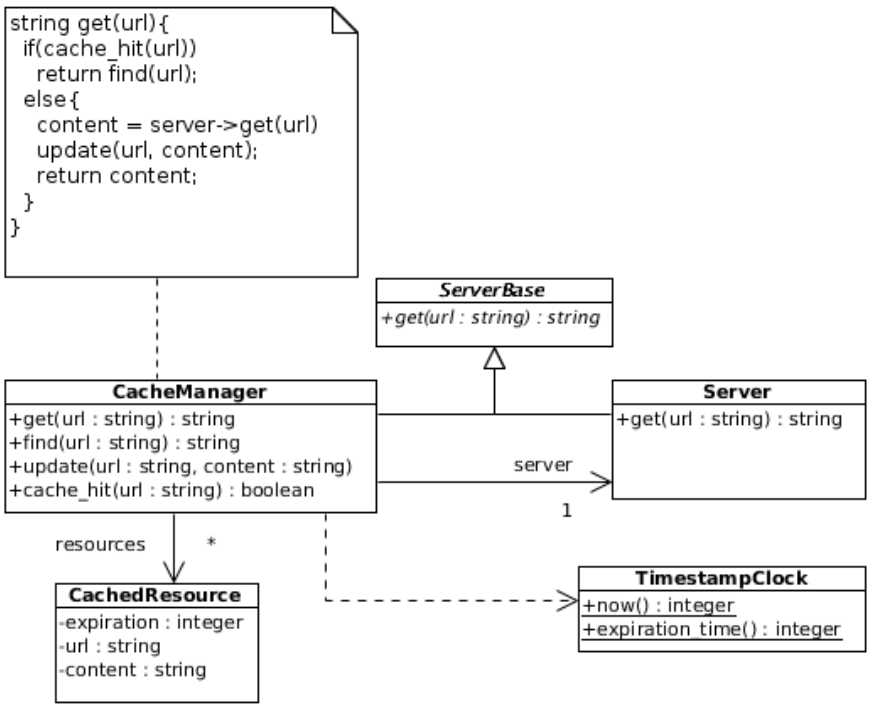
## **Problema 3** (30 puntos)

Durante el desarrollo de un servidor web se decide incorporar un sistema de cache con el objetivo de disminuir la carga del mismo. Para ello, se desea almacenar las respuestas que contienen recursos estáticos, para mejorar el rendimiento de futuras consultas.

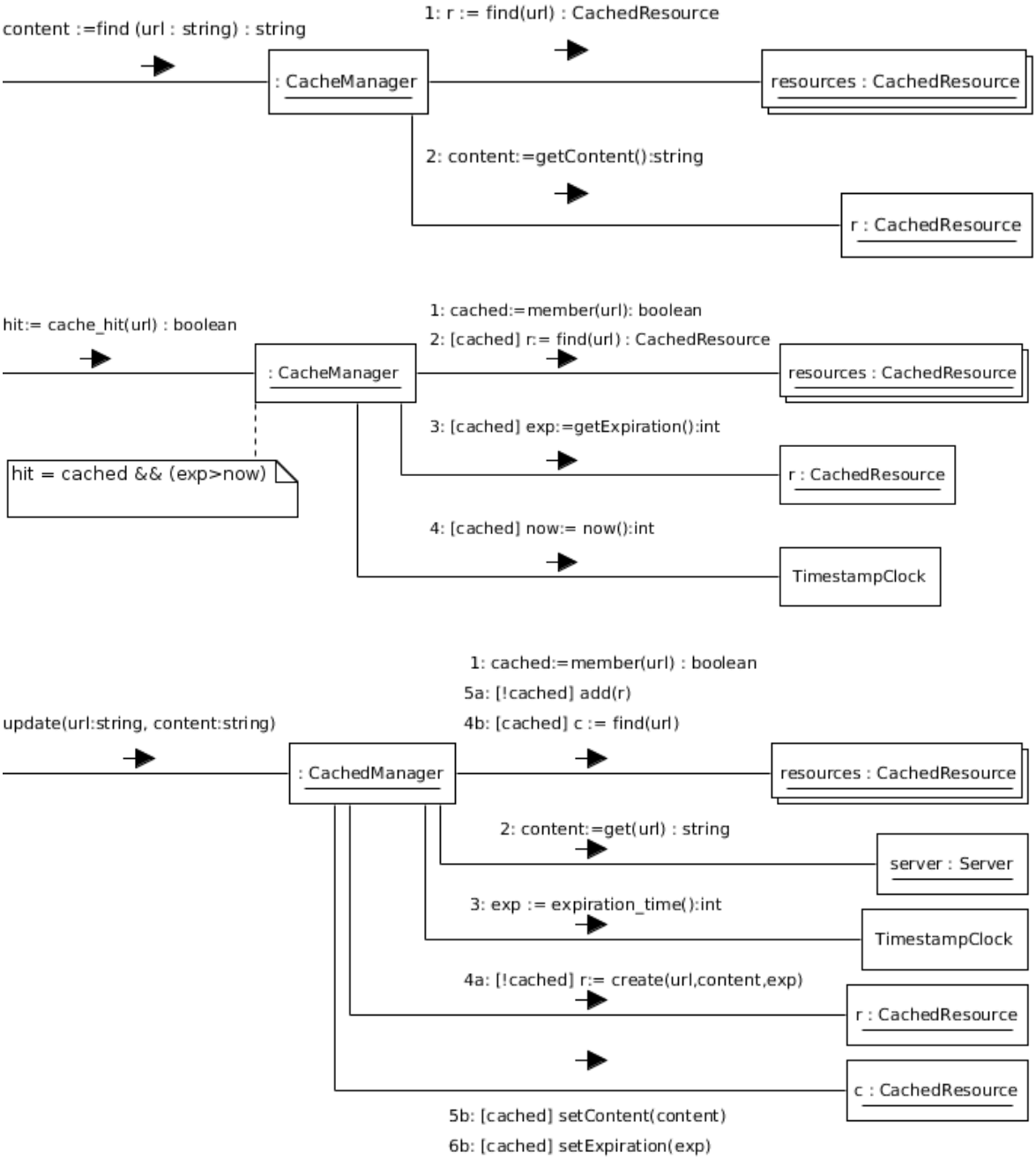
La clase **CacheManager** implementa la misma interfaz que la clase **Server**, y va a estar encargada de controlar el acceso al mismo. Devuelve el recurso si fue almacenado previamente y aún no expiró, o accede al **Server** para obtener el recurso y lo almacena en caso contrario.

En una etapa previa del proyecto se definió que para cada recurso se va a guardar su url (que lo identifica), el contenido que es de tipo string, y un timestamp que indica la fecha de expiración de dicho recurso. La fecha de expiración debe ser consultada antes de devolver el resultado almacenado previamente, el cual debe ser actualizado en caso de que la fecha actual sea mayor a la misma.

Con el objetivo de facilitar el manejo de timestamps, se cuenta con la clase **TimestampClock**, que provee la operación *now* que devuelve un timestamp de la fecha actual, y la operación *expiration_time* que devuelve un timestamp de la fecha de expiración para cualquier recurso almacenado al momento de ser invocado. A continuación se observa el DCD resultante.



A su vez, se generaron los siguientes **diagramas de comunicación** correspondientes a las operaciones de **CacheManager**.




**Se pide**

1. Implementar completamente en C++ las clases **CacheManager** y **ServerBase**. No es necesario incluir las directivas del preprocesador; asumir la existencia de implementaciones de IDictionary, IKey, IIterator y la interfaz ICollectible en su solución.
2. En caso de que existan, identificar los patrones de diseño utilizados en la solución. Para cada caso, indicar nombre y roles participantes (o en su defecto la colaboración completa de los mismos).