Programación Avanzada

EXAMEN DICIEMBRE 2008

19/12/2008

SOLUCIÓN

Problema 1 (Total: 28 puntos) [Teórico]

Responda las siguientes preguntas:

1. ¿Qué significan los conceptos *Herencia, Full Descriptor y Segment Descriptor* y cómo se relacionan? ¿Cómo se construye el Full Descriptor de un objeto?

Herencia es el mecanismo por el cual se permite compartir propiedades entre una clase y sus descendientes.

Un *Full Descriptor* es la descripción completa que es necesaria para describir a un objeto. Contiene la descripción de todos los atributos, operaciones, métodos y asociaciones que el objeto contiene.

Un *Segment Descriptor* son los elementos que efectivamente se declaran en un modelo o en el código (por ejemplo, clases) y contienen las propiedades heredables que son:

- Los atributos
- Las operaciones y los métodos
- La participación en asociaciones (los pseudoatributos)

El mecanismo de Herencia, define la forma en que el Full Descriptor de una clase es generado:

- Si un clase no tiene ningún padre entonces su Full Descriptor coincide con su Segment Descriptor.
- Si tiene uno o más padres, entonces su Full Descriptor se construye como la unión de su propio Segment Descriptor con los de todos sus ancestros.

La clase para la cual se genera el Full Descriptor hereda las propiedades especificadas en los segmentos de sus ancestros.

EXTRA: Para una clase no es posible declarar un atributo u operación con el mismo prototipo es más de uno de los segmentos. Si esto ocurriera el modelo estaría mal formado.

2. Durante el curso se utilizaron algunos diagramas de UML que permiten expresar interacciones dinámicas. Describa cada uno explicando:

Diagrama de Secuencia del Sistema:

i. En etapa de análisis, lo usamos para definir las interacciones que se dan entre los actores de un caso de uso y el sistema. Sirve para definir las operaciones

que va a tener el sistema, para posibilitar el envío de mensajes que permitan la realización de los casos de uso.

ii. Nos basamos en las especificaciones de los casos de uso.

iii. Hay:

- Actores: son los participantes en el caso de uso que da origen al DSS, y son quienes disparan eventos en el sistema, a través de los mensajes que envían.
- Objeto de Clase *"sistema"*: sistema es una clase que se utiliza para representar al sistema bajo análisis como un único objeto, capaz de recibir mensajes a través de sus operaciones.
- Mensajes: muestran mensajes que los actores le envían al sistema. Permite identificar las operaciones del sistema. El orden de envío es de arriba hacia abajo.

```
{objeto := } mensaje ({parámetros de entrada}){:TipoRetorno}
```

- Guardas: permiten alterar el flujo de los DSS:
  - Iteración (loop)
  - Condicionales
  - "include" de otros segmentos de otros DSS

Diagrama de Comunicación:

i. En etapa de diseño, permiten especificar el comportamiento del sistema ante los diversos estímulos (mensajes) que éste recibe.

ii. Para su elaboración, es necesario haber identificado las operaciones del sistema (DSS) y haber elaborado sus contratos, y también es importante tomar como guía la estructura planteada en el modelo de dominio. El diseño de las interacciones, se hace asignando responsabilidades a las clases, tomando como guía los criterios GRASP.

iii. Contienen:

- Objetos y Clases: representan instancias de objetos que están presentes en el sistema, a los cuales se les ha asignado alguna responsabilidad, necesaria para cumplir con el contrato de la operación en cuestión.
- Multiobjetos: colecciones de varios objetos.
- Links: unen los objetos, y permiten el envío de mensajes.
- Mensajes: permiten identificar las operaciones de las clases, tanto de instancia como de clase.
  - `#{*{[condición]}} : {[guarda]} mensaje`
  - El mensaje tiene la misma sintaxis que en el DSS
  - # refleja el orden y la anidación de los mensajes.
  - `*[condición]` indica iteración (y la condición para que esta siga, o no, realizándose)
  - `[guarda]` indica que el envío de mensaje, esta "atado" a una condición que se debe cumplir.
  - Además, para cada mensaje, se indica su sentido a través de una flecha que permite determinar el emisor y receptor del mensaje.
    - Se pueden agrupar todos los mensajes que tengan un mismo sentido, con una sola flecha.
  - El punto de entrada del DC, es un mensaje especial, que no incluye las partes de numeración, iteración y guardas. Es el que da inicio a la interacción.

- Visibilidad: para indicar si un objeto ve a otro porque el último es atributo del primero, o si el último es parámetro o es creado localmente, o es accesible por el resto de los objetos (global).

3. Describa el algoritmo usado (en el curso) para construir el *Diagrama de Clases de Diseño*. Nombre los artefactos previos que utiliza como entrada de este algoritmo (no es necesario definirlos).

a) Identificar todas las clases que participan de la solución de los casos de uso. Hacer esto analizando los diagramas de comunicación.
b) Incluirlas en el diagrama de clases.
c) Replicar los atributos de los conceptos correspondientes en el Modelo de Dominio, agregando aquellos nuevos que sean necesarios.
d) Agregar las operaciones correspondientes a cada clase analizando los diagramas de comunicación.
e) Agregar la información de tipos a los atributos y operaciones.
f) Agregar las asociaciones necesarias para permitir las visibilidades por atributo requeridas en los diagramas de comunicación.
g) Agregar navegabilidades para indicar la dirección de cada visibilidad por atributo.
h) Agregar dependencias para reflejar los demás tipos de visibilidades existentes.
i) Agregar interfaces, fabricas y datatypes.

4. ¿Qué significa (teóricamente) *Clase Abstracta* y qué elementos de **C++** son necesarios para implementar este tipo de Clases? Escriba un ejemplo, en **C++**, de una Clase Abstracta, y de otra Concreta, que le permita explicar todos los conceptos involucrados. No es necesario realizar la explicación, salvo que lo considere muy necesario.

Una Clase Abstracta es una clase que no se puede instanciar (no se pueden crear objetos que sean instancia directa de ella). Existen para que otras clases hereden sus propiedades.

En C++ es necesario recurrir a las funciones "virtuales puras", que son aquellas que pueden ser sobre-escritas y que no tienen método asociado.

```
virtual t_ret operación (l_param) = 0;
```

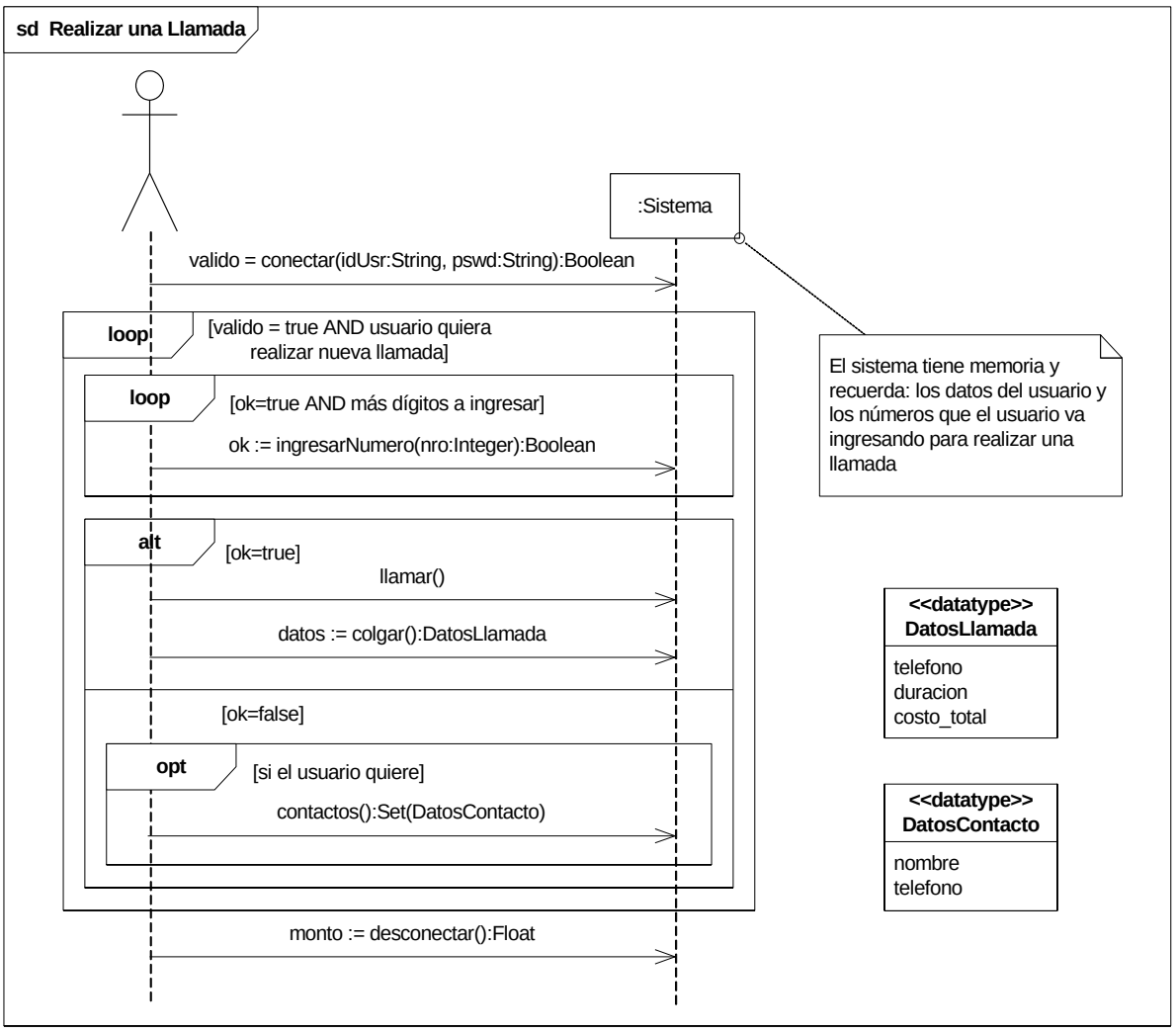
Una clase (c++) que tenga una de estas funciones no puede ser instanciada, y por lo tanto es abstracta.

```
/* una clase abstracta */
class Absrtracta {
      private:
            // Los dos atributos sirven para ilustrar
            // la herencia de atributos y asociaciones
            int atr;
            UnaClase* pseudoAtr;

      public:
            virtual ~Absracta(){};

            // funcion virtual pura que hace que
            // la clase sea abstracta
```

```
            virtual void unaOperacion() = 0;
};


/* una clase concreta */
class Concreta : public Abstracta {
      public:
            // implementacion de la fc. virtual pura ...
            virtual void unaOperacion(){
                  if (pseudoAtr != NULL)
                        // ... que involucra uso de
                        // propiedades heredables
                        atr = pseudoAtr->opEntera();
            }
};
```

Problema 2 (Total: 35 puntos) [Práctico]

i. Realice un único Diagrama de Secuencia del Sistema para el caso de uso anterior, incluyendo toda la información contenida en el mismo.



ii. Especifique las pre- y post- condiciones de la operación *altaContacto()*.

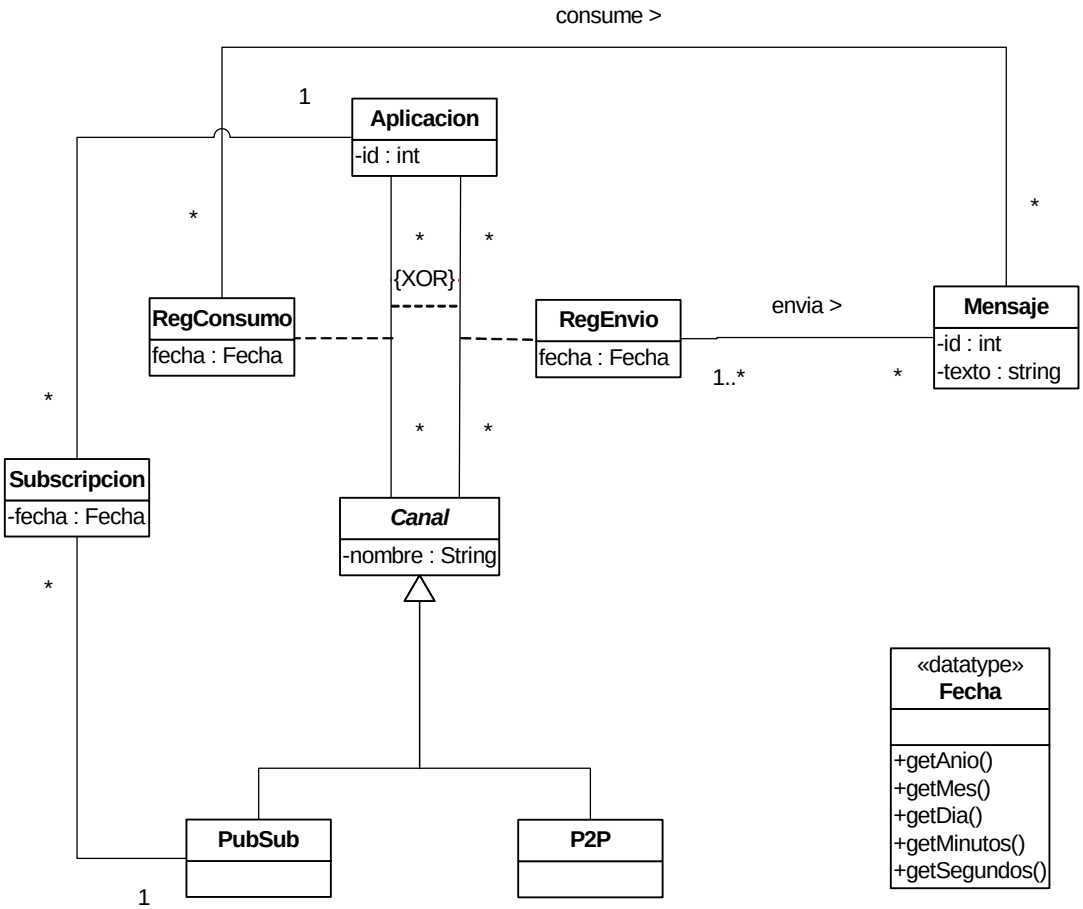
Operación	**altaContacto(ID:String, nomC:String, numC:Integer):Boolean**
Pre- y poscondiciones	
pre: existe un objeto de la clase Usuario con idUsr = ID (esta pre puede obviarse si se considera que la sesión está abierta, i.e., el usuario existe) **post:** si el objeto Contacto con nombre = nomC no existe, se crea una instancia de Contacto con nombre = nomC y numero = numC; y se genera un link entre esa instancia y la instancia de Usuario con idUsr = ID **post:** si la instancia de Contacto con nombre = nomC ya existe entonces se devuelve false, en caso contrario se devuelve true	

Problema 3 (Total: 35 puntos) [Práctico]

Se pide:

Construya un Diagrama de Modelo e Dominio UML para la realidad descripta. Las restricciones deben ser expresadas en lenguaje natural. Modele exclusivamente en base a la información presente en la descripción.

Modelo



Restricciones

```
-- El identificador de la aplicación es único.

-- El identificador del mensaje es único.

-- El nombre del canal es único.

-- Un mensaje consumido por una aplicación debe haber sido previamente enviado
por otra aplicación.

-- Un mensaje enviado por un canal P2P puede ser consumido por a lo sumo por
una aplicación.

-- Una aplicación subscripta a un canal pub/sub tiene un registro de consumo
sobre dicho canal y la fecha de subcripción es mayor a la fecha del registro
de consumo.

-- Los registros de envío de un mensaje corresponden a la misma aplicación.
```