Programación Avanzada

PRIMER PARCIAL – 23/10/2008

SOLUCIÓN

_____ _____

Nombre y Apellido c.i.

Por favor siga las siguientes indicaciones:

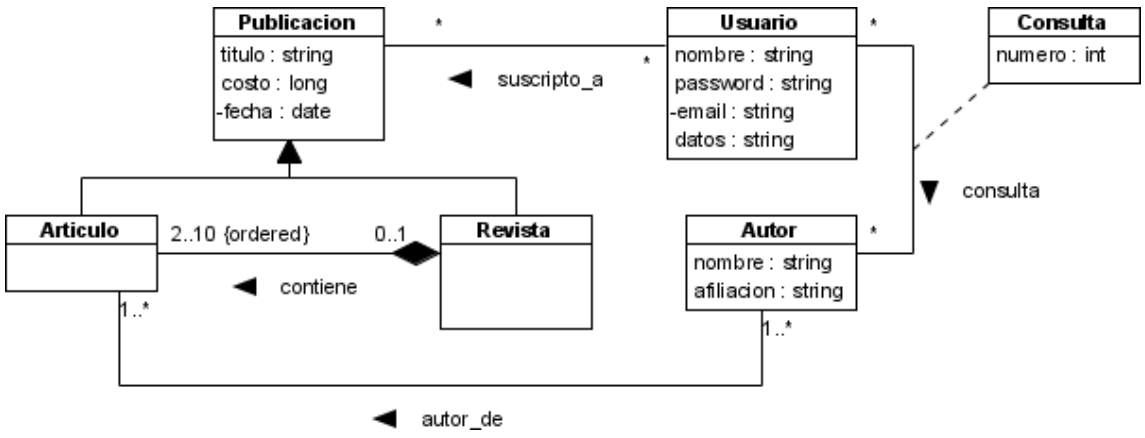
- Escriba con lápiz.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Escriba las hojas de un solo lado.
- Comience cada ejercicio en una hoja nueva.
- El total máximo de puntos del parcial es 100.

Problema 1 (Total: 60 puntos) [Práctico]

Se está desarrollando un sitio web con un sistema de suscripción a publicaciones electrónicas. Una descripción reducida del dominio y del caso de uso principal aparece a continuación.

Se pide:

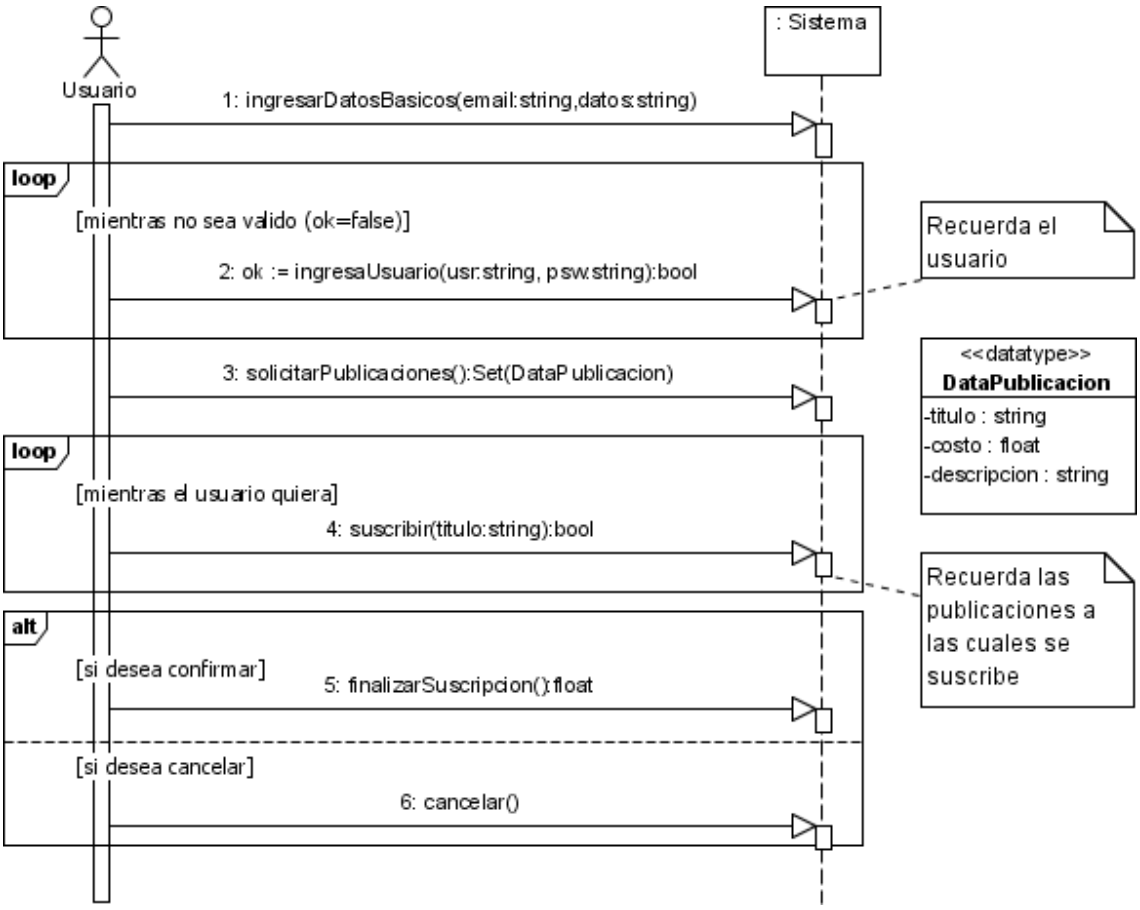
i. (35 puntos) Construir el Modelo de Dominio para la realidad descripta y presentarlo en un diagrama utilizando UML. Las restricciones deberán ser expresadas en lenguaje natural. Modelar exclusivamente en base a la información presente en la descripción y los casos de uso.



Restricciones

```
-- El título de la publicación es único.

-- El nombre del usuario es único.

-- La fecha de publicación de un artículo es la misma de la
-- revista en la que aparece

-- El costo de una revista es siempre menor o igual a la
-- suma del costo de los artículos que contiene

-- Un usuario no puede estar al mismo tiempo suscripto a un
-- artículo y a la revista que lo contiene
```

ii. (25 puntos) Realice <u>un único</u> Diagrama de Secuencia de Sistema (DSS) para el caso de uso *"Suscripción a Publicaciones"*, incluyendo <u>toda</u> la información contenida en el mismo.



Problema 2 (Total: 40 puntos) [Teórico]

Responda brevemente cada una de las siguientes preguntas:

1. (10 puntos) Conceptos básicos de Orientación a Objetos
   a) Indicar y definir cuáles son las posibles relaciones entre dos conceptos de un Modelo de Dominio

Asociación: relación que describe una relación semántica entre dos clasificadores.
Generalización: relación entre un elemento más general y un elemento más específico. El elemento más específico es consistente con el mas general y puede tener información adicional.

   b) Defina "tipo asociativo" haciendo especial énfasis en las multiplicidades de éste.

Es un elemento del que es clase y asociación al mismo tiempo. Permite agregar propiedades a las asociaciones.
Respecto a las multiplicidades: existe una única instancia del tipo asociativo por cada par de instancias de los conceptos que asocia.

   c) ¿Qué es una agregación compuesta?

Es una agregación en la que las partes son exclusivas del compuesto. Además, en general, una acción sobre el compuesto se propaga sobre las partes.

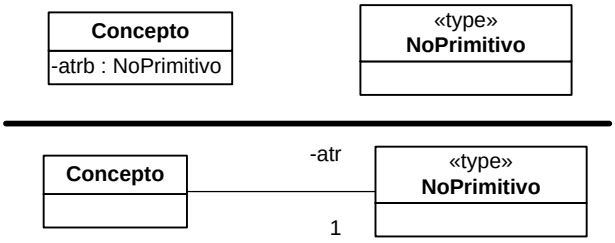
   d) ¿Cuál es la utilidad del rol en las asociaciones?

Los roles son necesarios para eliminar la ambigüedad sobre la asociación a la que se hace referencia cuando existe más de una asociación entre dos clases.

2. (10 puntos) Modelado del dominio
   a) Describir las técnicas de identificación de conceptos vistas en el curso.

Lista de categorías de conceptos: consiste en repasar las lista de categoría de conceptos buscando los conceptos del dominio del problema que apliquen a cada categoría.
Identificación de sustantivos: se identifican los sustantivos de una descripción textual del problema y se los considera como conceptos o atributos candidatos.

   b) Muestre, mediante un ejemplo sencillo, cómo se define utilizando UML un tipo no primitivo y un atributo cuyo tipo es el tipo no primitivo definido anteriormente.



3. (10 puntos) Especificación del Comportamiento
   a) ¿Qué especifica y como está estructurado un *Contrato de Software?*.

Un contrato de software especifica el comportamiento o efecto de una operación. Se estructura especificando un nombre para la operación, las responsabilidades de la misma, el tipo del cual la operación es propiedad, las referencias cruzadas (caso/s de uso a los que pertenece la operación), notas generales, salida de la operación, las pre y post-condiciones, y opcionalmente, snapshots que ejemplifiquen el estado de la instancia a la se le aplicó la invocación, previo y posterior a la invocación.

   b) ¿Qué significa el concepto "*memoria del sistema*" en el contexto de los DSS's?

Son los datos que el Sistema debe guardar temporalmente mientras se este ejecutando un Caso de Uso. Representa el estado de la conversación entre el usuario y Sistema.

   c) Explique la relación entre caso de uso, escenario y diagrama de secuencia del sistema.

Para cada Caso de Uso pueden existir varios escenarios (típicos y alternativos). Cada escenario de un Caso de Uso se puede representar con un DSS.

4. (10 puntos) Elementos básicos de Implementación
   a) Fundamentar el objetivo del operador *virtual* (en lo que refiere a herencia).

Redefinición de funciones en clases derivadas: En una clase derivada se puede definir una función que ya existía en la clase base, esto se conoce como "overriding", o superposición de una función.

Funciones virtuales: Cuando en una clase declaramos una función como virtual, y la superponemos en alguna clase derivada, al invocarla usando un puntero de la clase base, se ejecutara la versión de la clase derivada.
Una vez que una función es declarada como virtual, lo seguirá siendo en las clases derivadas, es decir, la propiedad virtual se hereda.
Si la función virtual no se define exactamente con el mismo tipo de valor de retorno y el mismo número y tipo de parámetros que en la clase base, no se considerará como la misma función, sino como una función superpuesta.

b) En *c++:* ¿cómo se implementa una operación abstracta? ¿Y una clase abstracta? Dé un ejemplo sencillo.

Funciones virtuales puras: Una función virtual pura es aquella que no necesita ser definida. El modo de declarar una función virtual pura es asignándole el valor cero.

Sintaxis:

```
virtual <tipo> <nombre_función>(<lista_parámetros>) = 0;
```

Clases abstractas: Una clase abstracta es aquella que posee al menos una función virtual pura. No es posible crear objetos de una clase abstracta, estas clases sólo se usan como clases base para la declaración de clases derivadas.
Las funciones virtuales puras serán aquellas que siempre se definirán en las clases derivadas, de modo que no será necesario definirlas en la clase base.

EJEMPLO:

```
// Vehiculo.hh
// Es la clase base, y es abstracta porque posee una
// operación abstracta (getCarga()).
// Las funciones miembro están definidas en Vehiculo.cc

#ifndef Vehiculo_HH
#define Vehiculo_HH

class Vehiculo {
private:
      int codigo;
      int viajesPorSemana;

public:
      // constructor
      Vehiculo(int,int);

      // destructor
      virtual ~Vehiculo();

      // setter's
      void setCodigo(int);
      void setViajesPorSemana(int);

      // getter's
      int getCodigo();
      int getViajesPorSemana();

      // devuelve la carga por semana del vehiculo
      float getCapacidadCarga();

      // abstracta, cada una de las clases que derivan de vehiculo
      // implementan sus respectivos metodos.
      virtual float getCarga()=0;
};

#endif
```

```
// Camion.hh
// Clase derivada de Vehiculo.
// Las funciones miembro están definidas en Camion.cc

#ifndef CAMION_HH_
#define CAMION_HH_

#include "Vehiculo.hh"

class Camion: public Vehiculo {
private:
      int cantEjes;
      float cantKgPorEje;

public:
      // Camion (c,v,cE,kgE) crea un nuevo objeto, donde 'c'
      // es el codigo del camion, 'v' la cantidad de viajaes
      // por semana, 'cE' es la cantidad de ejes y 'kgE' es
      // la cantidad de kg por ejes.
      Camion(int,int,int,float);

      // destructor
      ~Camion();

      // setter's
      void setCantEjes(int);
      void setCantKgPorEje(float);

      // getter's
      int getCantEjes();
      float getCantKgPorEje();

      // devuelve la carga maxima del camion
      float getCarga();
};

#endif
```

Problema 3 (sin puntaje) [Laboratorio]
(Este problema no es parte de la evaluación parcial por lo que no se le asignan puntos. Sin embargo, es parte de la evaluación del laboratorio con el fin de evaluar si el estudiante participó activamente en el mismo. Por dicha razón, es obligatoria su realización.)

1. Implementar en C++ el archivo de cabecera de la clase Fecha. No es necesario incluir directivas al preprocesador en el código.

```
// Fecha.hh
// Las funciones miembro están definidas en Fecha.cc

class Fecha {
  private:
     int dia;  // 1-30
     int mes;  // 1-12
     int anio; // cualquier año

  public:
     // **CONSTRUCTORES**
     // Constructor por defecto.
     Fecha ();

     // Constructor por copia.
     // Fecha(f) construye una Fecha copia de f.
     Fecha (const Fecha &);

     // Constructor comun.
     // Fecha(d,m,a) construye una Fecha a partir de los datos
ingresados.
     Fecha (int,int,int);

     // **DESTRUCTOR**
     // Libera la memoria utilizada por la instancia.
     virtual ~Fecha();

     // **GET**
     // Retorna el dia del objeto implicito
     int getDia() const;

     // Retorna el mes del objeto implicito
     int getMes() const;

     // Retorna el anio del objeto implicito
     int getAnio() const;

     // **SET**
     // Setea el dia del objeto implicto
     void setDia(int);

     // Setea el dia del objeto implicto
     void setMes(int);

     // Setea el dia del objeto implicto
     void setAnio(int);

     // **OPERADORES**
     // operator=(f) Asigna al objeto implicito la
     // fecha referenciada por f
```

```
    Fecha& operator=(const Fecha &);

    // operator+(dd) Suma al objeto imlpicito d dias.
    Fecha operator+(const int);

    // operator-(dd) Resta al objeto implicito d dias.
    Fecha operator-(const int);

    // operator==(f) Retorna true sii el objeto implicito es
    // igual a f
    bool operator==(const Fecha &) const;

    // operator!=(f) Retorna true sii el objeto implicito es
    // distinto a f
    bool operator!=(const Fecha &) const;

    // operator<(f) Retorna true sii el objeto implcito es menor
    // a la fecha f.
    bool operator<(const Fecha &) const;

    // operator>(f) Retorna true sii el objeto implcito es mayor
    // a la fecha f.
    bool operator>(const Fecha &) const;

    // operator>=(f) Retorna true sii el objeto implcito es mayor
    // o igual a la fecha f.
    bool operator>=(const Fecha &) const;

    // operator<=(f) Retorna true sii el objeto implcito es menor
    // o igual a la fecha f.
    bool operator<=(const Fecha &) const;

};

   // **SOBRECARGAS DE LOS OPERADORES DE INSERCCION Y EXTRACCION DE
   // FLUJO**
   // Escribe una fecha en un flujo cualquiera.
   ostream& operator<<(ostream &, const Fecha &);

   // Lee una fecha de un flujo cualquiera.
   istream& operator>>(istream &, Fecha &);

   Fecha operator+(const int ,const Fecha &);

   Fecha operator-(const int dd,const Fecha &f);
```

2. En el contexto del laboratorio (Sistema de Gestión de Reclamos), realice el Contrato de Software del Caso de Uso *Cancelar Reclamo.*

Nombre	**Cancelar reclamo**
Operación	`bajaReclamo(codReclamo:int)`
Entrada	codReclamo es el codigo del reclamo que se quiere cancelar.
Salida	No hay.
Descripción	Cancela el reclamo con codigo codReclamo existente en el sistema si el mismo no tiene horas registradas.
Excepciones	Si no se cumple pre1 se lanza la excepción ExNoExisteReclamo. Si no se cumple pre2 se lanza la excepción ExReclamoConDedicacion.
Precondiciones y Postcondiciones	
Pre1: No existe ninguna instancia del tipo asociativo TareaReclamo que linkea al reclamo con codigo = codReclamo con alguna tarea. **Pre2: Existe en el sistema una instancia de Reclamo con codigo = codReclamo** **Post1: Se elimino el link entre el reclamo con codigo = codReclamo y el tipo de articulo que el reclamo tenia asociado y posteriormente se elimino del sistema la instancia del reclamo indicada.**	