Programación Avanzada

SEGUNDO PARCIAL – 14/07/2009

_____ _____

Nombre y Apellido C.I.

Por favor, lea atentamente y siga las siguientes indicaciones:

- El parcial contiene un total de: **6** páginas.
- Escriba con lápiz.
- Al finalizar el parcial deberá entregarse esta hoja con todos los datos identificatorios. 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.
- Las preguntas del problema 1 son de múltiple opción, su respuesta debe darse en los casilleros de abajo. Cada una de esas preguntas bien contestada vale: **4,5** puntos, mal contestada: **-1,5** puntos, no contestada: **0** punto. Escriba con claridad, respuestas dudosas se consideraran no contestadas.
- El total máximo de puntos del parcial es **60**.
- La duración del parcial es de 3:00 horas.

Respuestas a los ejercicios de Múltiple Opción:

	Pregunta 1:	Pregunta 2:	Pregunta 3:	Pregunta 4:
PROBLEMA 1				

	Problema 1:	Problema 2:	Problema 3:	Total:
Solo para uso docente				

Problema 1 (Total: 18 puntos) [MO]

En las siguientes preguntas, solo una opción es correcta.

1. ¿Cómo es posible crear objetos de una clase cuyos constructores son todos privados?

   a) Definiendo un método estático público en la clase que cree un objeto de la clase y lo devuelva.
   b) Definiendo una subclase y declarando públicos los constructores heredados.
   c) Definiendo una superclase con constructores públicos.
   d) No es posible

2. Cuando se emplea el paso por referencia para un parámetro de una función:

   a) Se invoca al constructor de copia.
   b) Se invoca al constructor de copia sólo si el parámetro también se ha definido como *const*.
   c) No se invoca al constructor de copia.
   d) No se invoca al constructor de copia sólo si el parámetro también se ha definido como *const*.

3. Una clase puede tener:

   a) Todos los constructores que se desee.
   b) Sólo el constructor por defecto y el constructor de copia.
   c) Un destructor por cada constructor.
   d) Las anteriores respuestas no son correctas.

4. Respecto a la herencia:

   a) La parte privada no es heredada por las clases derivadas.
   b) La parte protegida de una clase sólo es accesible por los métodos de la propia clase.
   c) Una relación "tiene-un" se implementa mediante la herencia pública.
   d) Las anteriores respuestas no son correctas.

Problema 2 (Total: 10 puntos) [Práctico]

Dada las siguientes definiciones de clases:

```
//-------------------
// CLASE BASE
//-------------------
class cBase {
      private:
            int objeto_base_1;
            int objeto_base_2;

      public:
            cBase();
            virtual ~cBase();
            void funcion_A();
            void funcion_B();
            virtual void funcion_C();
            virtual void funcion_D() = 0;
};

cBase::cBase():   objeto_base_1(1), objeto_base_2(2) {
      cout << "Constructor de BASE" << endl;
}

cBase::~cBase() {
      cout << "Destructor de BASE" << endl;
}

void cBase::funcion_A() {
      cout << "Funcion A de BASE" << endl;
}

void cBase::funcion_B() {
      cout << "Funcion B de BASE" << endl;
}

void cBase::funcion_C() {
      cout << "Funcion C de BASE" << endl;
}


//-------------------
// CLASE DERIVADA
//-------------------
class cDerivada : public cBase {
      private:
            int objeto_deriv_1;
            int objeto_deriv_2;

      public:
            cDerivada();
            ~cDerivada();
            void funcion_A();
            void funcion_D();
};
```

```
cDerivada::cDerivada(): cBase(), objeto_deriv_1(3), objeto_deriv_2(4) {
      cout << "Constructor de DERIVADA" << endl;
}

cDerivada::~cDerivada() {
      cout << "Destructor de DERIVADA" << endl;
}

void cDerivada::funcion_A() {
      cout << "Funcion A de DERIVADA" << endl;
}

void cDerivada::funcion_D() {
      cout << "Funcion D de DERIVADA" << endl;
}
```

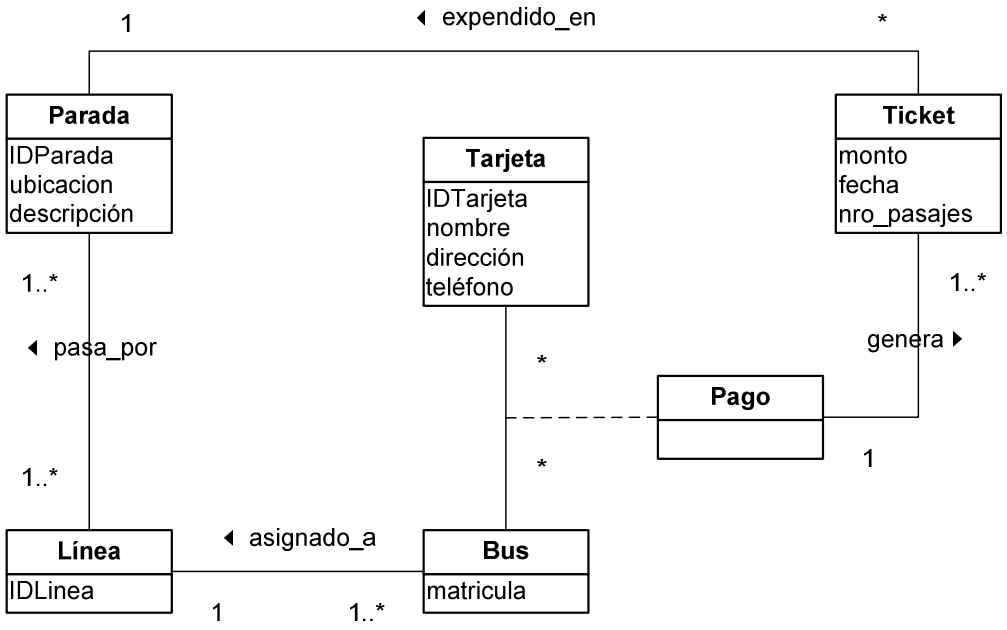
SE PIDE:

Suponer que se ejecuta el programa. Hacer una lista enumerando ordenadamente cada operación invocada a raíz de dicha ejecución, justificando. Recordar que una línea de código puede causar que se invoque más de una operación.

```
1  int main() {
2
3      cDerivada derivada;
4      cBase *base_ptr = &derivada;
5
6      derivada.funcion_A();
7      base_ptr->funcion_A();
8
9      derivada.funcion_B();
10     base_ptr->funcion_B();
11
12     derivada.funcion_C();
13     base_ptr->funcion_C();
14
15     derivada.funcion_D();
16     base_ptr->funcion_D();
17
18 }
19
```

Problema 3 (Total: 32 puntos) [Práctico]

La Intendencia Municipal de Montevideo (IMM) desea informatizar el sistema de transporte urbano, incluyendo, entre otros, un servicio de débito automático para los usuarios del transporte capitalino. El equipo de desarrollo generó el siguiente Modelo de Dominio con la información relativa al sistema de gestión de débito central de la IMM.



NOTAS:

1) IDParada, IDTarjeta, IDLinea y matricula identifican a las paradas, tarjetas, líneas y buses respectivamente.
2) La parada en que se expende un ticket debe corresponder con algunas de las paradas de la línea del bus en el que se paga el ticket.

Actualmente se está trabajando con el caso de uso de *Débito de Línea* encargado de dar de alta la información de los tickets expendidos en los ómnibus de una línea cada cierto período y consultar el monto recaudado en cada parada. El caso de uso es resuelto por las operaciones cuyos contratos se muestran a continuación.

Nombre	**Alta Tickets de Bus**
Operación	`altaTicketsDeBus(dT:Set[DataTicket], bus:matricula)`
Entrada	*dT* – DataType compuesto por el monto, fecha y números de pasajes expendidos a un pasajero de una sola vez; el identificador de la tarjeta del pasajero y el identificador de la parada donde se expendió el ticket *bus* – matrícula del bus para el cual se cargan los datos
Salida	Ninguna
Descripción	Da de alta los Ticket expendidos en un Bus determinado en el sistema de gestión central de la IMM
Precondiciones	El bus, las paradas y las tarjetas de cada ticket existen en el sistema
Postcondiciones	Se dan de alta los tickets para el bus determinado. Se generan las asociaciones que se crean convenientes.

Nombre	**Consultar Recaudación en Parada de una Línea**
Operación	`consultarRecaudacion(idP:Integer, idL:Integer):Float`
Entrada	*idP* – identificador de la Parada a consultar *idL* – identificador de la Línea a consultar
Salida	Monto recaudado
Descripción	Devuelve el monto recaudado en cierta parada para una línea determinada
Precondiciones	La parada y la línea existen en el sistema
Postcondiciones	Se devuelve el monto de todos los tickets expendidos en la parada indicada para la línea indicada

SE PIDE:

Realice la colaboración (Diagrama de Comunicación y Diagrama de Clases de Diseño) del caso de uso *Débito de Línea* teniendo en cuenta que no se puede modificar el Modelo de Dominio. Nombre, además, las visibilidades existentes (no es necesario explicitarlas en los diagramas).