Programación Avanzada

SEGUNDO PARCIAL – 4/12/2008

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**.
- El parcial contiene un total de: **4** páginas.

Problema 1 (Total: 40 puntos) [Teórico]

Responda brevemente cada una de las siguientes preguntas:

a) Nombrar 3 (tres) criterios GRASP y explicar brevemente en qué consiste cada uno.

Ver juego de diapositivas *"Diseño – Criterios de Asignación de Responsabilidades"*.

b) ¿Qué 2 (dos) enfoques se citaron en el curso para diseñar una colaboración?

1) Definir primero la estructura y luego generar las diferentes interacciones respetando dicha estructura.
2) Definir las interacciones (teniendo en cuenta los criterios GRASP) y luego definir la estructura necesaria para que dichas interacciones puedan ocurrir.

c) ¿Qué tipos de de visibilidad pueden existir entre objetos, en el contexto de un diagrama de comunicación, y cuándo se dan estas visibilidades?

Existen 4 formas básicas de que un objeto A tenga visibilidad sobre otro B:

- por atributo (o asociación): B es un pseudo-atributo de A.
- por parámetro: B es un parámetro de un método de A.
- local: B es declarado localmente en un método de A.
- global: B es visible de forma global.

d) Describir las responsabilidades que pueden ser asignadas a una colección de objetos en un diagrama de comunicación.

Las colecciones son tratadas como meros contenedores de objetos por lo que no tendrán otra responsabilidad más que esa.
Proveen solamente operaciones que permitan administrar los objetos contenidos.
En general las interfaces **Diccionario** (add, remove, find y member) e **Iterador** (next) son suficientes para las colecciones.

e) ¿Qué es un controlador?

Un controlador es una clase que implementa las operaciones del sistema.

f) ¿Qué relación existe entre controladores e interfaces del sistema?

Las interfaces del sistema contienen las operaciones del sistema, las cuales son implementadas por los controladores.

g) ¿Qué elementos están presentes en los diagramas de clases de diseño y no en los diagramas de modelo de dominio?

En el diagrama de clases de diseño aparecen: **dependencias, navegabilidades, realizaciones, interfaces** y **operaciones.**

h) Defina brevemente como se mapean a C++ las relaciones de generalización, realización y dependencia.

Generalización: en la declaración de la clase se especifica su ancestro.
```
class Jornalero : public Empleado
```

Realización: en la declaración de la clase se especifica la(s) interfaz(ces) que realiza. En C++ se utiliza una generalización.
```
class ATM : IRetiro
```

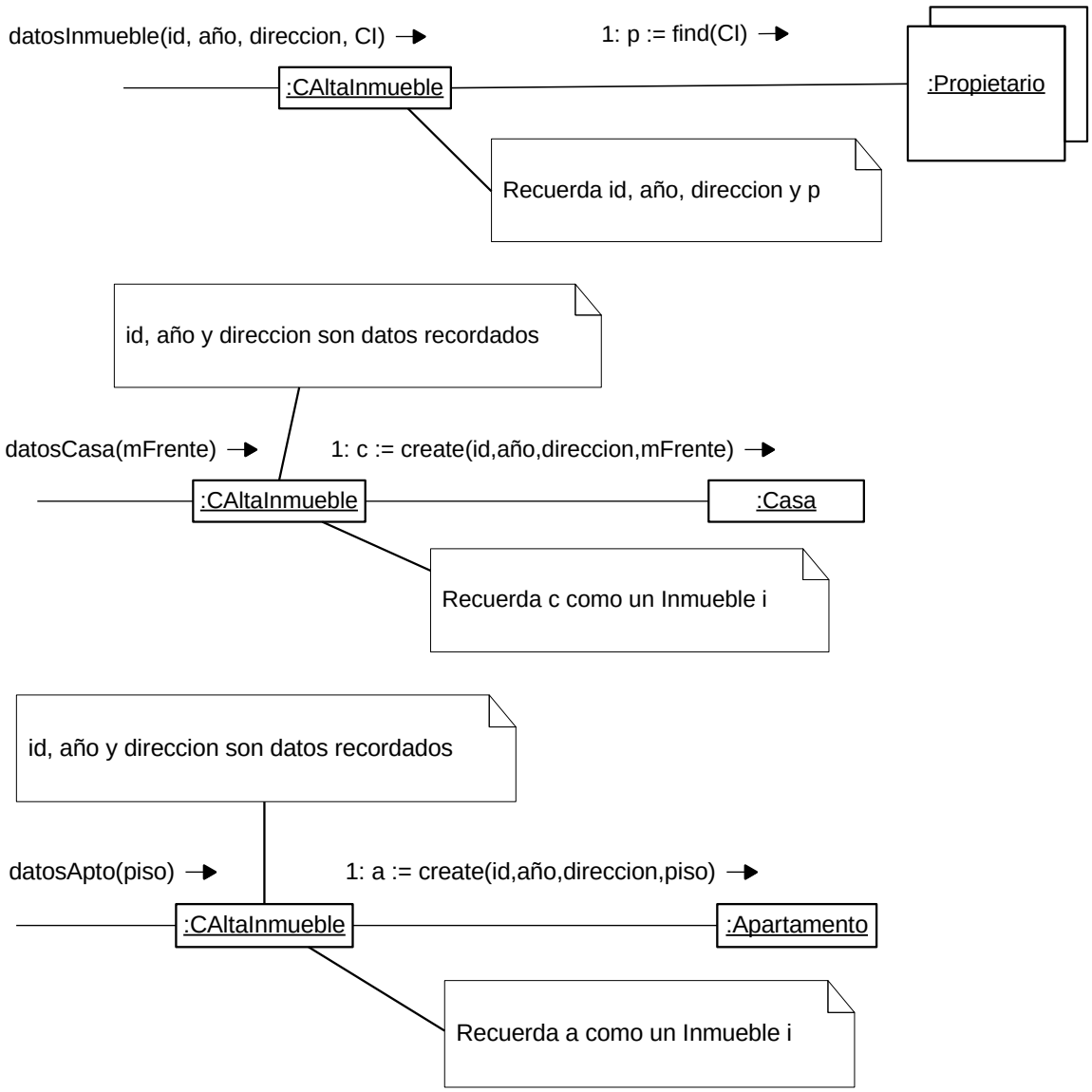
Dependencia: Las dependencias se declaran en la definición de un elemento para tener visibilidad sobre otros. Una asociación navegable, una generalización y una realización son también formas de dependencia.
En C++ se utiliza `#include`

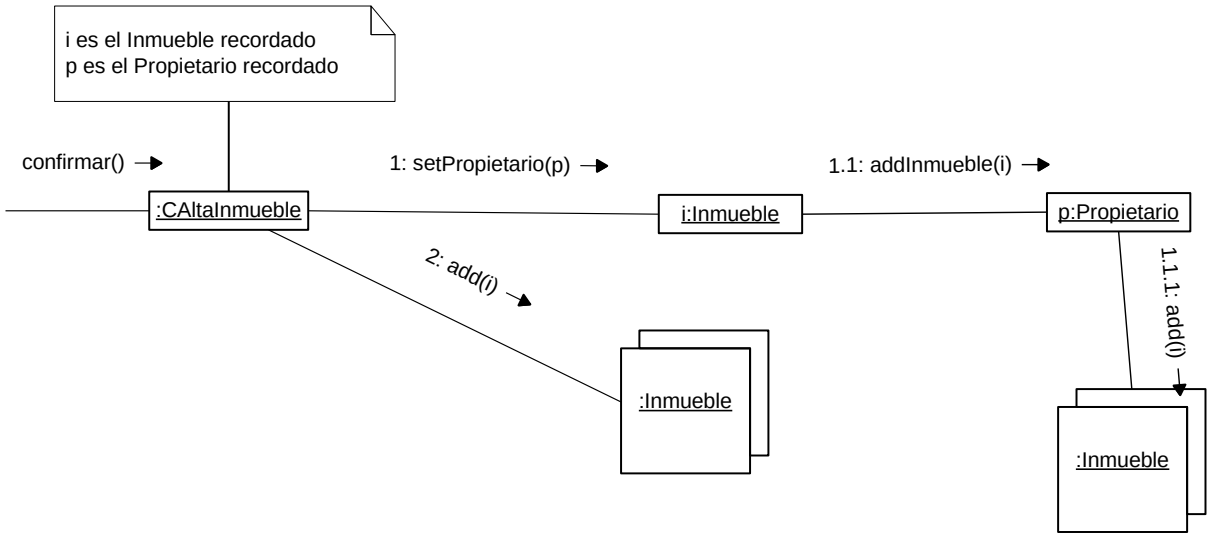
i) Explique brevemente que es una colección genérica y una colección concreta (o tipada).

Colecciones concretas (templates): el tipo del elemento a almacenar es declarado como parámetro que será instanciado el generar la colección particular.
Colecciones genéricas: pueden almacenar directamente cualquier tipo de elemento.

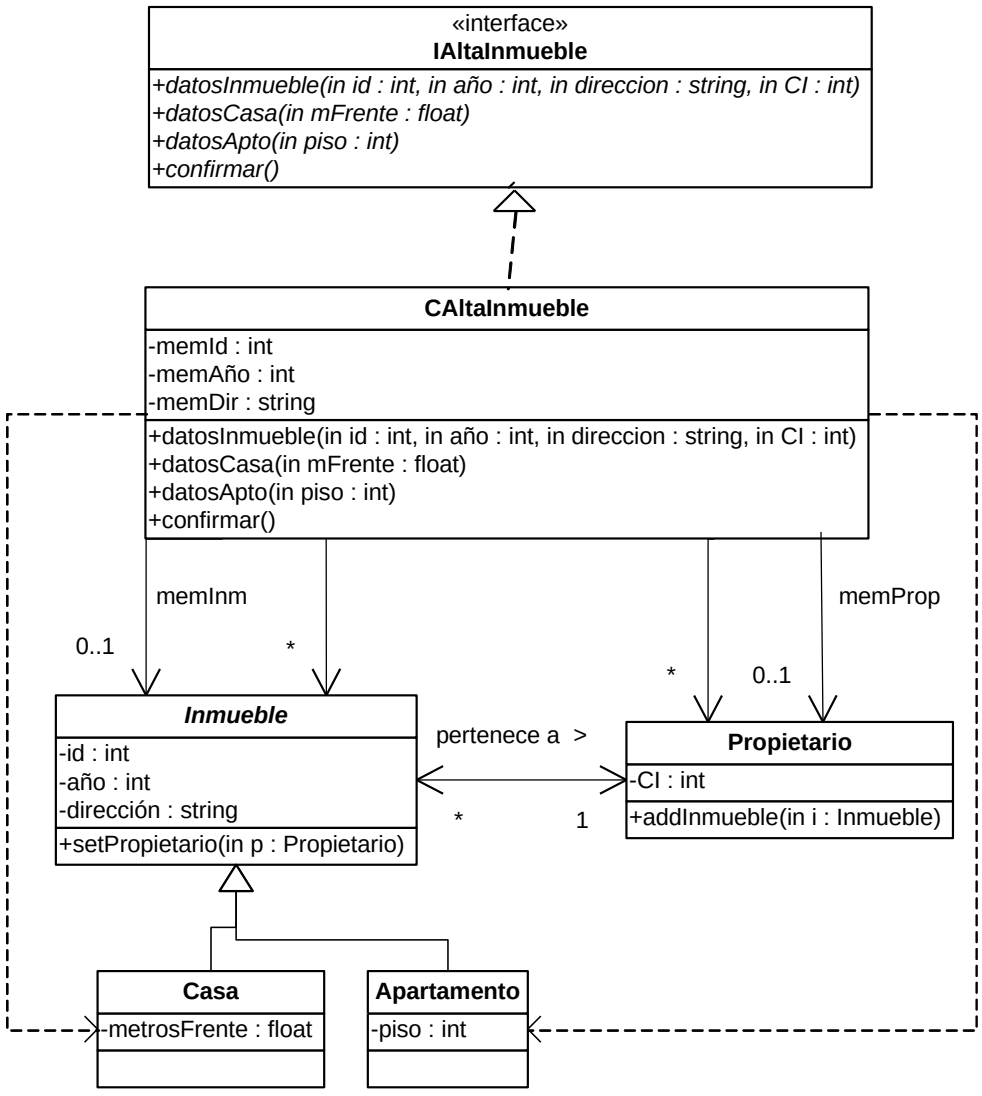
Problema 2 (Total: 40 puntos) [Práctico]

PARTE **A**:
i.




i es el Inmueble recordado
p es el Propietario recordado
confirmar()
1: setPropietario(p)
1.1: addInmueble(i)
:CAltaInmueble
i:Inmueble
p:Propietario
2: add(i)
1.1.1: add(i)
:Inmueble
:Inmueble

ii.


«interface»
IAltaInmueble
+datosInmueble(in id : int, in año : int, in direccion : string, in CI : int)
+datosCasa(in mFrente : float)
+datosApto(in piso : int)
+confirmar()
CAltaInmueble
-memId : int
-memAño : int
-memDir : string
+datosInmueble(in id : int, in año : int, in direccion : string, in CI : int)
+datosCasa(in mFrente : float)
+datosApto(in piso : int)
+confirmar()
memInm
memProp
0..1
*
*
0..1
Inmueble
-id : int
-año : int
-dirección : string
+setPropietario(in p : Propietario)
pertenece a >
*
1
Propietario
-CI : int
+addInmueble(in i : Inmueble)
Casa
-metrosFrente : float
Apartamento
-piso : int

PARTE **B:**

```
class IAltaInmueble {
  public:
   virtual void datosInmueble(int id, int anio, String direccion, int CI) = 0;
   virtual void datosCasa(float mFrente) = 0;
   virtual void datosApto(int piso) = 0;
   virtual void confirmar() = 0;
   virtual ~IAltaInmueble();
};

class CAltaInmueble : public IAltaInmueble {
      private:
            IDictionary *inmuebles, *propietarios;
            int memId, memAnio;
            String memDir;
            Inmueble *memInm;
            Propietario *memProp;
            CAltaInmueble();
            static CAltaInmueble *instancia;
      public:
            void datosInmueble(int id, int anio, String direccion, int CI);
            void datosCasa(float mFrente);
            void datosApto(int piso);
            void confirmar();
            static CAltaInmueble *getInstancia();
};

CAltaInmueble *CAltaInmueble::instancia = NULL;

CAltaInmueble *CAltaInmueble::getInstancia() {
      if (instancia == NULL)
            instancia = new CAltaInmueble;
      return instancia;
}

CAltaInmueble::CAltaInmueble() {
      inmuebles = new ListDictionary();
      propietarios = new ListDictionary();
}

CAltaInmueble::datosInmueble(int id, int anio, String direccion, int CI) {
      memId = id;
      memAnio = anio;
      memDir = direccion;
      memProp = propietarios->find(CI);
}

CAltaInmueble::datosCasa(float mFrente) {
      memInm = new Casa(memId, memAnio, memDir, mFrente);
}

CAltaInmueble::datosApto(int piso) {
      memInm = new Apartamento(memId, memAnio, memDir, piso);
}

CAltaInmueble::confirmar() {
      memInm->setProp(memProp);
      inmuebles->add(memInm);
}

class Inmueble : public ICollectible {
      private:
            int id, anio;
```

```
            String direccion;
            Propietario *prop;
      public:
            Inmueble(int id, int anio, String direccion);
            void setPropietario(Propietario *prop);
            int getKey();
            virtual ~Inmueble();
};

class Casa : public Inmueble {
      private:
            float mFrente;
      public:
            Casa(int id, int anio, String direccion, float mFrente);
};

class Apartamento : public Inmueble {
      private:
            int piso;
      public:
            Apartamento(int id, int anio, String direccion, int piso);
};

class Propietario : public ICollectible {
      private:
            int CI;
            IDictionary *inmuebles;
      public:
            Propietario(int CI);
            int getKey();
            void addInmueble(Inmueble *i);
};
```

Problema 3 (Total: 20 puntos) [Práctico]

PARTE A:

1	void main(){	
2	Clase c1;	Constructor por defecto
3	Clase c2(3,2.54);	Constructor comun
4	Clase c3=c2;	Constructor copia
5	c1=c2+c3;	*Constructor copia *Operator+ *Constructor copia (resultado de operator+)(copia temporal que es la que se usa como parametro de operator=) *Constructor copia (parametro de operator=) *operator= *Constructor copia (resultado de operator=)
6	c1.desplegar();	Metodo desplegar
7	}	Destructor

PARTE B:

1	claseA *a	Tipo Estatico: claseA
2	claseB *b	Tipo Estatico: claseB
3	claseC *c	Tipo Estatico: claseC
4	claseD *d	Tipo Estatico: claseD
5	claseA *ap	Tipo Estatico: claseA
6	claseB *bp	Tipo Estatico: claseB
7	int in = 0	
8	short sh = 4	
9		
10	a = new claseA()	Tipo Dinamico: claseA
11	b = new claseB()	Tipo Dinamico: claseB
12	c = new claseC()	Tipo Dinamico: claseC
13	d = new claseD()	Tipo Dinamico: claseD
14		
15	ap = b	Tipo Dinamico: claseB
16	bp = c	Tipo Dinamico: claseC
17		
18	ap->func1(in)	Metodo de la claseA
19	ap->func3(in)	Metodo de la claseB
20	bp->func3(in)	Metodo de la claseC
21	bp->func1(in)	Metodo de la claseC
22	bp->func2(in)	Metodo de la claseB
23		
24	ap = c	Tipo Dinamico: claseC
25	ap->func2(in)	Metodo de la claseB
26	ap->func2(sh)	Metodo de la claseC
27		
28	ap = b	Tipo Dinamico: claseB
29	ap->func2(sh)	Metodo de la claseA
30		
31	ap = d	Tipo Dinamico: claseD
32	ap->func1(in)	Metodo de la claseA
33	a->func1(in)	Metodo de la claseA
34		
35	bp = b	Tipo Dinamico: claseB
36	bp->func2(sh)	Metodo de la claseA