Programación 4

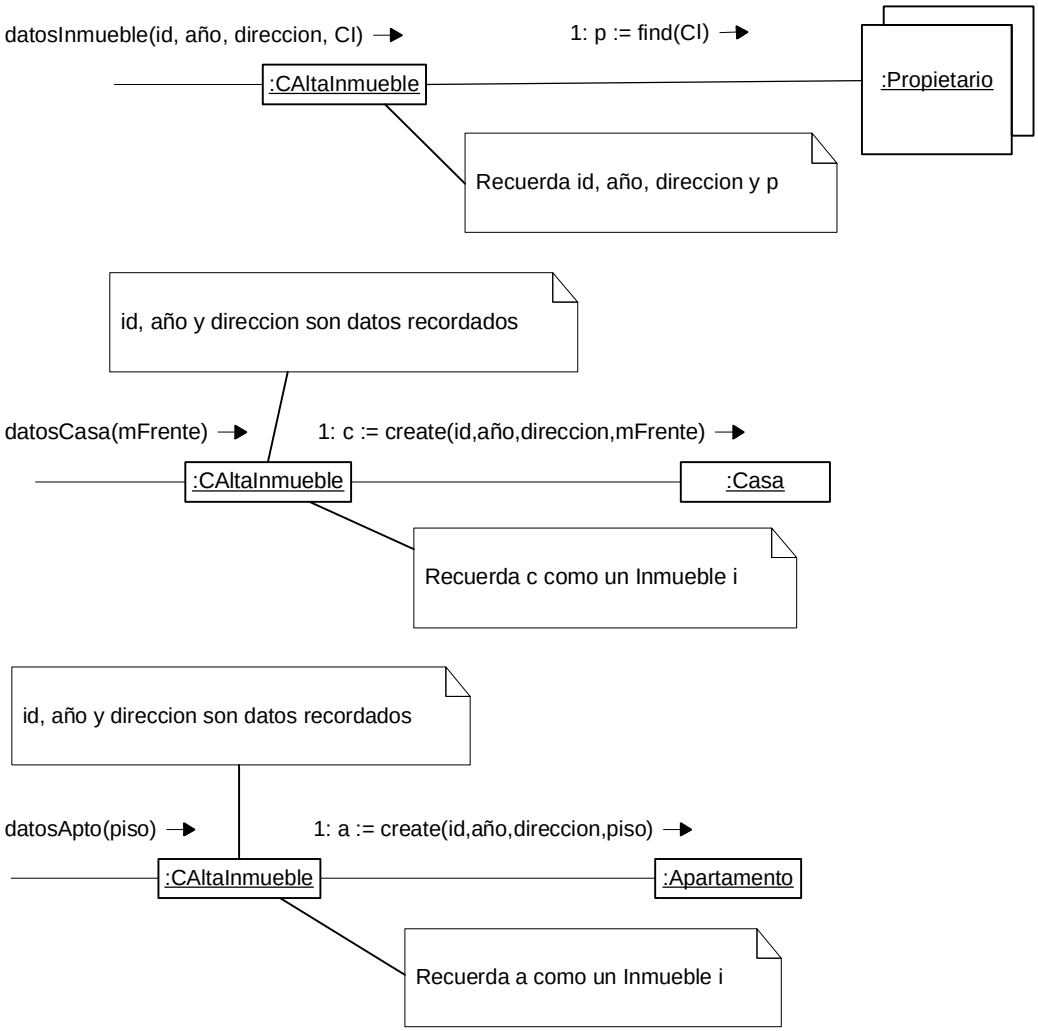
EXAMEN JULIO 2008

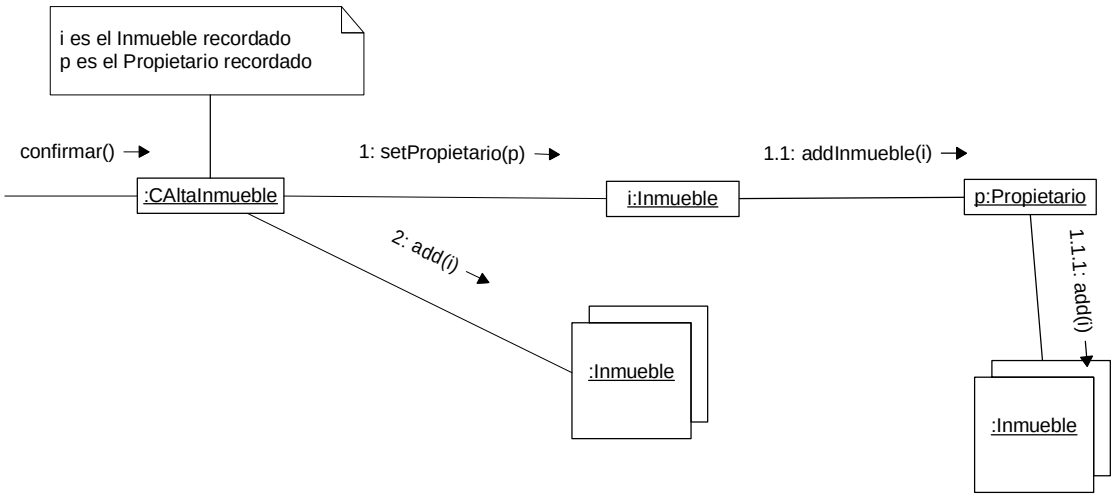
SOLUCION

Problema 1

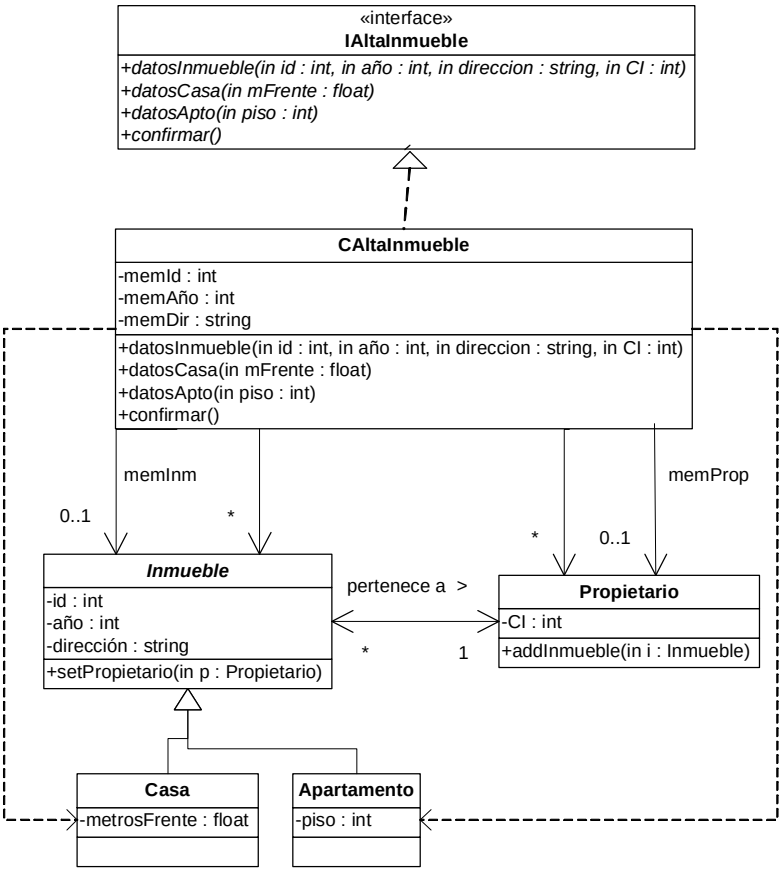
PARTE A:

i.





ii.



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);
}
```

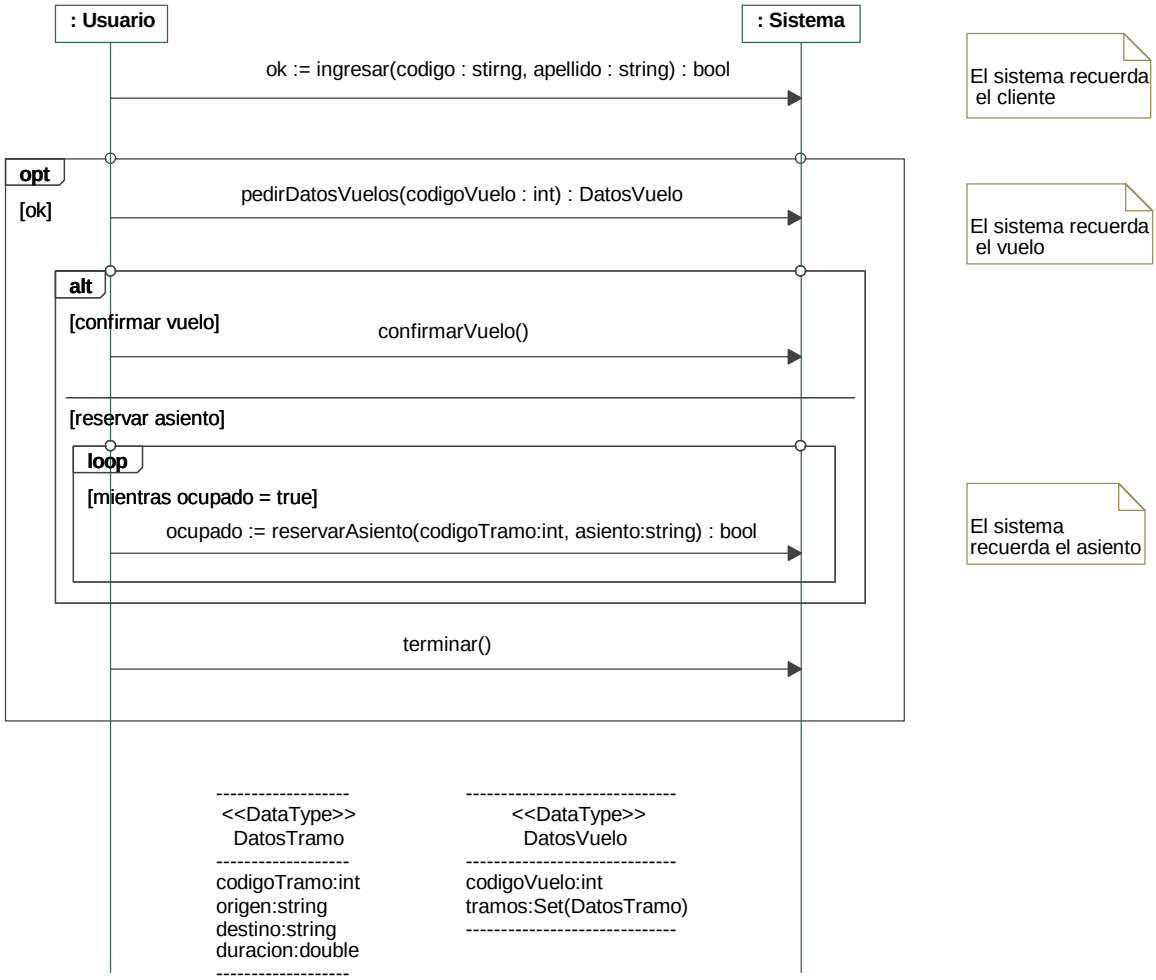
Problema 2

a) ¿Qué significa que en la etapa de Análisis se trata al Sistema como una "caja negra"?

Significa que no se conoce cómo funciona el sistema internamente, sólo se conoce su interfaz.

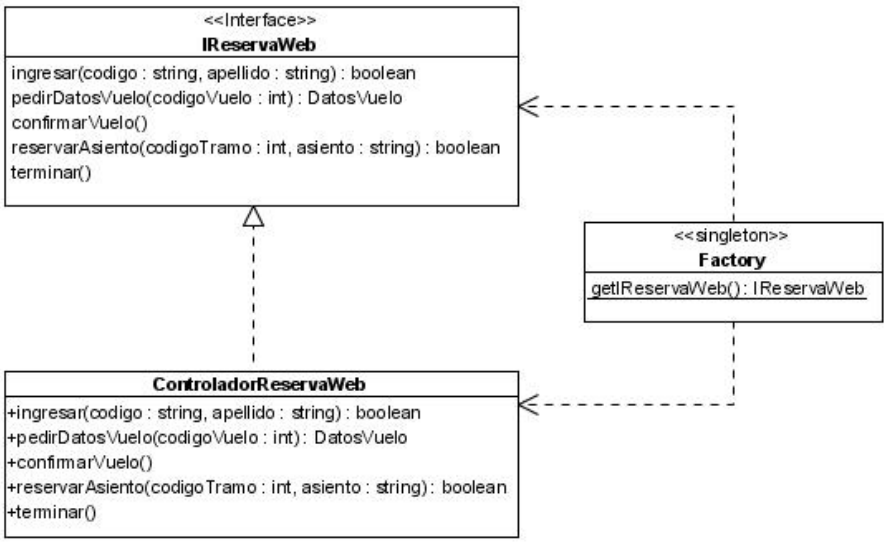
b)

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



ii. Defina el/los controladores así como la/las interfaces y fábricas necesarias, justificando su opción, y muestre un diagrama de clases conteniéndolos. No incluya el resto de las clases que podrían ser parte del diseño del caso de uso.

Dado que existe un caso de uso complejo (que necesita un sistema con memoria) se optó por utilizar un único controlador de caso de uso ControladorReservasWeb.



Problema 3

a)

i. Definir qué es una asociación e instancia de asociación. Indicar cómo se pueden categorizar las asociaciones de un Modelo de Dominio.

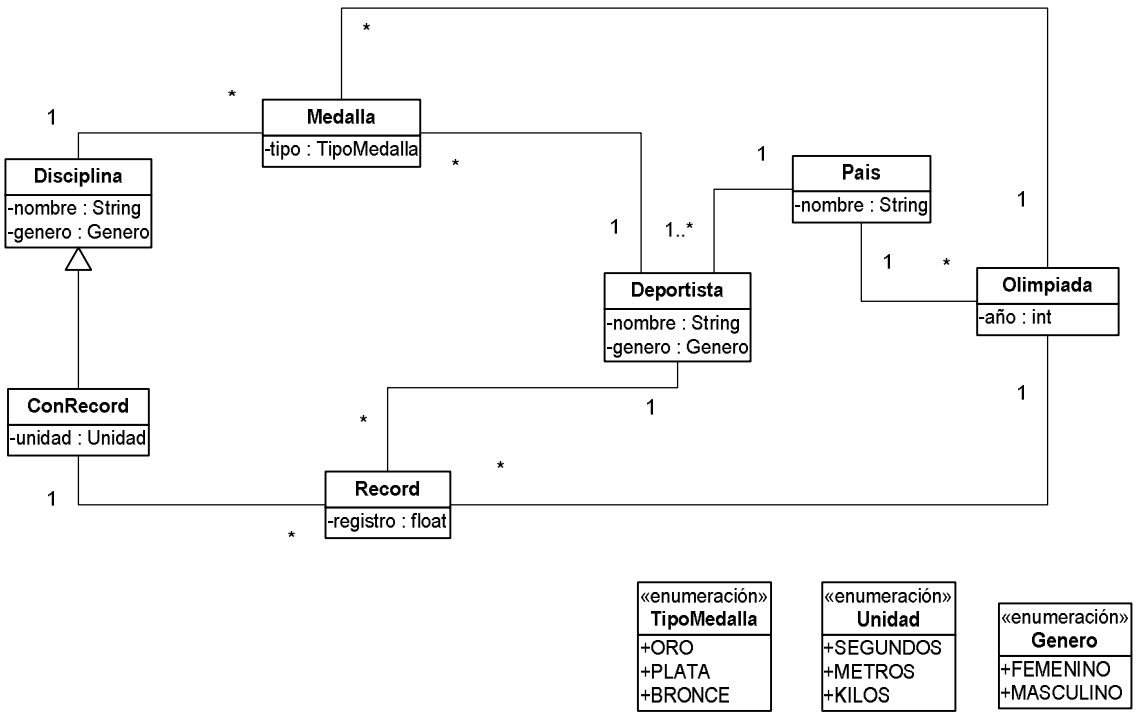
Una asociación es una relación entre conceptos que indica una relación interesante o significativa entre ellos. Una instancia de asociación es una tupla que relaciona instancias de clasificadores. Las asociaciones se pueden categorizar en asociaciones de comprensión y need-to-nkow.

ii. Definir qué es una restricción al Modelo de Dominio. Indicar las formas vistas para representar las restricciones y realizar una comparación entre ellas (ventajas y desventajas)

Una restricción al Modelo de dominio es un predicado que expresa una condición sobre los elementos del modelo y que siempre debe ser verdadero. Las restricciones se pueden representar en lenguaje natural (entendibles pero ambiguas) o mediante OCL (único significado pero más complejo y requiere aprendizaje).

b)

i. Modelar la realidad planteada mediante un Diagrama de Modelo de Dominio UML.



ii. Expresar las restricciones tanto en lenguaje natural como en OCL.

```
-- El nombre de la disciplina es único.
context Disciplina inv:
   Disciplina.allInstances()->isUnique(nombre)

-- El nombre de un deportista es único.
context Deportista inv:
   Deportista.allInstances()->isUnique(nombre)

-- Un deportista obtiene medallas de disciplinas de su género.
context Deportista inv:
   self.medalla.disciplina->forAll(d:Disciplina|d.genero =
   self.genero)

-- Un deportista registra récords de disciplinas de su género.
context Deportista inv:
   self.record.conrecord->forAll(d:Disciplina|d.genero =
   self.genero)

-- En una disciplina con récord de unidad segundos, los récords se
baten con un registro menor.
context ConRecord inv:
   self.unidad = Unidad::SEGUNDOS implies self.record->forAll(r1,
r2:Record| r1.registro < r2.registro implies r1.olimpiada.año
>=r2.olimpiada.año)

-- En una disciplina con récord de unidad metros o kilos, los
récords se baten con un registro mayor.
context ConRecord inv:
   (self.unidad = Unidad::METROS or self.unidad = Unidad::KILOS)
implies self.record->forAll(r1, r2:Record| r1.registro > r2.registro
implies r1.olimpiada.año >=r2.olimpiada.año)


-- Un deportista obtuvo alguna medalla o registra algún récord.
context Deportista inv:
self.record->size() > 0 or self.medalla->size() > 0
```