Tema 2: Lección 1

Ejercicio 1: Responde verdadero (V) o falso (F)

Dos objetos con el mismo estado pueden tener distinta identidad.	
Si hay mil objetos de una clase X habrá mil copias de su variable de clase x1.	
El código: **MuertoViviente vampiro;** crea en Java un objeto de la clase MuertoViviente.	
El código: **attr_writer :color** crea el get y el set del atributo color en Ruby.	
En Ruby los métodos de instancia son públicos y los atributos de instancia son privados, por defecto.	
La identidad de un objeto en programación orientada a objetos la da su dirección de memoria.	
Los constructores por defecto devuelven void.	
La encapsulación de un conjunto de elementos implica de forma implícita su ocultamiento para el resto de elementos del sistema.	
Cuando definimos una clase, definimos el estado y el comportamiento de un conjunto de objetos, y en algunas ocasiones también definimos estado y/o comportamiento de la propia clase.	
Todos los lenguajes de programación soportan los siguientes atributos de visibilidad de forma explícita para sus atributos: private , package y public	
Para invocar a los métodos de clase no es necesario que exista previamente una instancia de dicha clase en el sistema.	
Los paquetes son específicos de Java.	
En Java y Ruby las clases se tratan como objetos mientras que en C++ las clases constituyen un patrón	

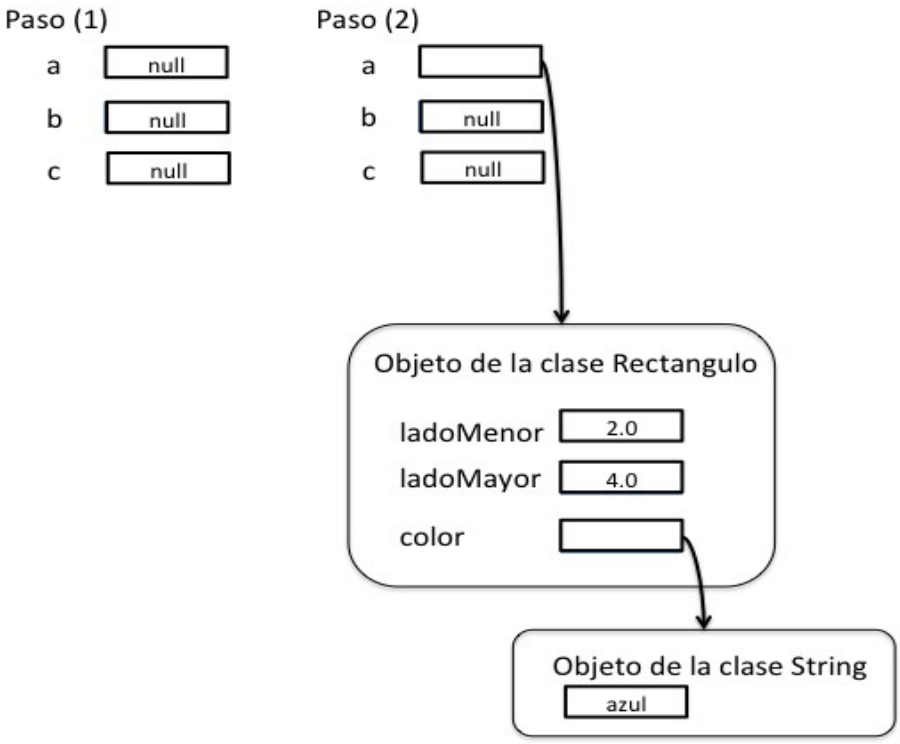
Ejercicio 2. Sea la clase java Rectángulo:

```
class  Rectangulo {
      float ladoMenor, ladoMayor;
      String color;

      Rectangulo(){
            ladoMenor=2;
            ladoMayor=4;
            color="azul";
      }
      float setColor(String unColor) {
            color = unColor;
      }
      float area(){
            return ladoMenor*ladoMayor;
      }
}
```

Representa lo que va ocurriendo con cada una de estas instrucciones. Se proporciona el resultado de los pasos (1) y (2), haz una figura para cada uno de los pasos (3), (4), (5) y (6).

```
(1)      Rectangulo a, b, c;
(2)      a = new Rectangulo();
(3)      b = new Rectángulo();
(4)      a.setColor("rojo");
(5)      float x = a.area();
(6)      c = a;
```



Ejercicio 3. Dada la clase Java:

```
public class Prueba {
      public static final int A = 1;
      static String s = "";
      private  int b= 2;
      int c;
}
```

a. ¿Cuántos atributos tiene? Indica si se trata de atributos de instancia o de clase. ¿Cuál es la visibilidad de cada uno de ellos?

b. A partir del siguiente código, suponiendo que está en otra clase del mismo paquete:

1. Prueba obj1 = new Prueba();
2. Prueba obj2 = new Prueba();
3. obj1.A = 3;
4. Prueba.s = "hola";
5. Prueba.A = 14;
6. obj1.b = 0;
7. obj2.b = 5;
8. obj1.c = 4;
9. obj2.c = 6;

b.1. ¿Se produciría algún error de compilación? ¿por qué? (Recomendación: ejecutar el código para comprobarlo).

b.2. Indica cuál es el estado de obj1 y obj2 después de su ejecución y eliminados los errores de compilación si los había.

Ejercicio 4. ¿Quién tiene la responsabilidad de responder a los mensajes que se corresponden con los métodos de clase?

Ejercicio 5. Razona si las siguientes afirmaciones son ciertas o falsas, suponiendo que estamos en la clase citada:

a) Los atributos de clase son accesibles sólo desde métodos de clase (no desde métodos de instancia)
b) Los atributos de instancia son accesibles sólo desde métodos de instancia (no desde métodos de clase).
c) La palabra reservada "this" (Java) puede emplearse tanto en métodos de clase como de instancia.

Ejercicio 6. ¿Qué mecanismos tiene Java para la ocultación de información?

Ejercicio 7.

Sea la clase C.java:

```
Class C {
   private static int contador2 = 0;
   private int contador1= 0;
   public void incrementarContador1() {contador1++;}
   public void incrementarContador2() {contador2++;}
   public int getContador1() {return contador1;}
   public int getContador2() {return contador2;}
}
```

Tras ejecutar el siguiente trozo de código:

```
C objeto1 = new C();
C objeto2 = new C();
objeto1.incrementaContador1();
objeto1.incrementaContador2();
objeto2.incrementaContador1();
objeto2.incrementaContador2();

int valor1 = objeto1.getContador1();
int valor2 = objeto1.getContador2();
int valor3 = objeto2.getContador1();
int valor4 = objeto2.getContador2();
```

a) ¿Qué valores tienen valor1, valor2, valor3 y valor4?

b) Traduce el código Java proporcionado a Ruby

Ejercicio 8.

Sean los ficheros A.java y B.java cuyo contenido es:

A.java

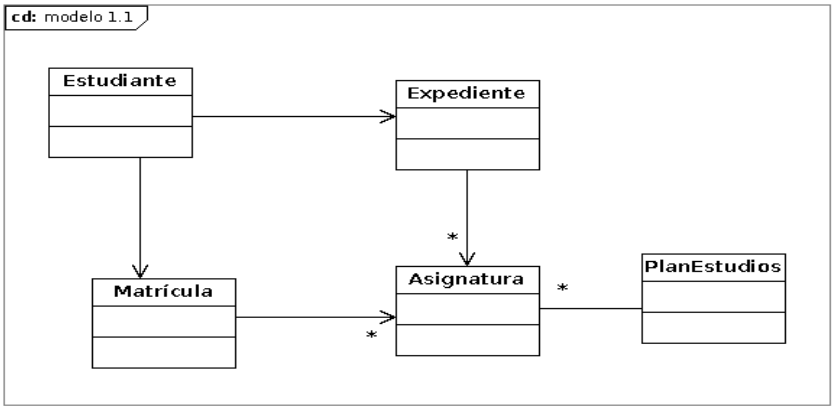
```
package p1;
import p2.B;
class A { (...) }
```

B.java

```
package p2;
public class B {
    String atributo1;
    public String atributo2;
   (…) }
```

Responde Verdadero (V) o Falso (F)	
La clase A pertenece al paquete p1	
Como se ha hecho un import de la clase B, atributo1 es accesible desde A	
Como se ha hecho un import de la clase B, atributo2 es accesible desde A	
Traduce a Ruby el código anterior	

Ejercicio 9. A partir del siguiente diagrama de clases, obtén un esquema en el que se muestren los objetos y sus enlaces (relaciones) para el siguiente caso: 2 objetos PlanEstudios, 6 objetos Asignatura, 5 objetos Estudiante. Haz las suposiciones que consideres oportunas.



Ejercicio 10. Las siguientes instrucciones en Java pretenden declarar Arrays. Haz pruebas en Java para entender cómo funcionan. Indica qué errores ves y escríbelas correctamente en Java.

	Motivo de error, si lo hay	Solución al error, en su caso
int [][] c;		
int [5] e;		
int d[];		
int f[]=new int[3];		

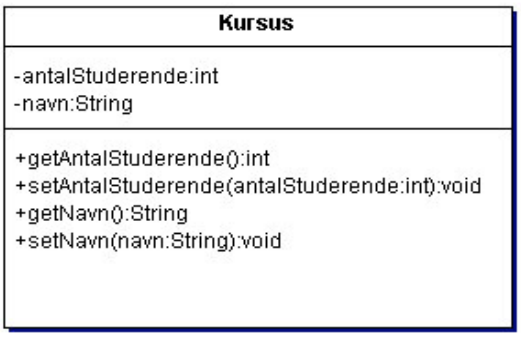
int[] x=new int[10];		
Array.newInstance(int, 5);		
int [] dims={2,4};		
Array.newInstance(Alumno, dims);		

Ejercicio 11. Define en Java y en Ruby una clase cuyas instancias representen atletas y otra clase cuyas instancias sean un equipo de atletas y un entrenador al frente. Incluye los atributos que consideres necesarios. Además, ten en cuenta que necesitamos saber cuántos equipos de atletas hay. Escribe un programa sencillo que cree un equipo y muestre

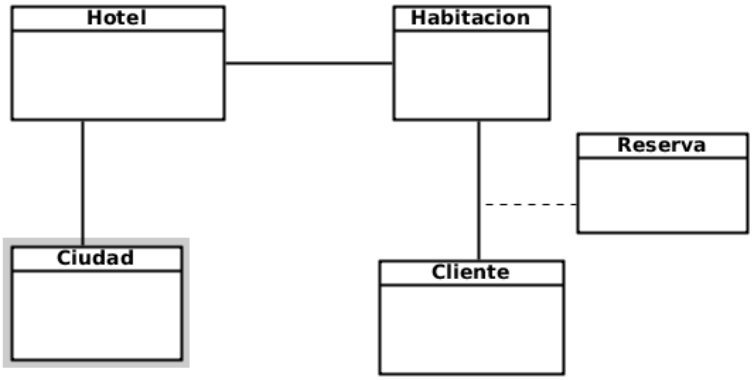
a) Los atletas que corren en el mismo y su entrenador.
b) El número de equipos que tenemos

Tema 2: Lección 2

Ejercicio 1. UML es un lenguaje “universal”. Escribe el código Java y Ruby correspondiente a la declaración de la siguiente clase en danés (incluyendo la declaración de atributos y la cabecera de los métodos).

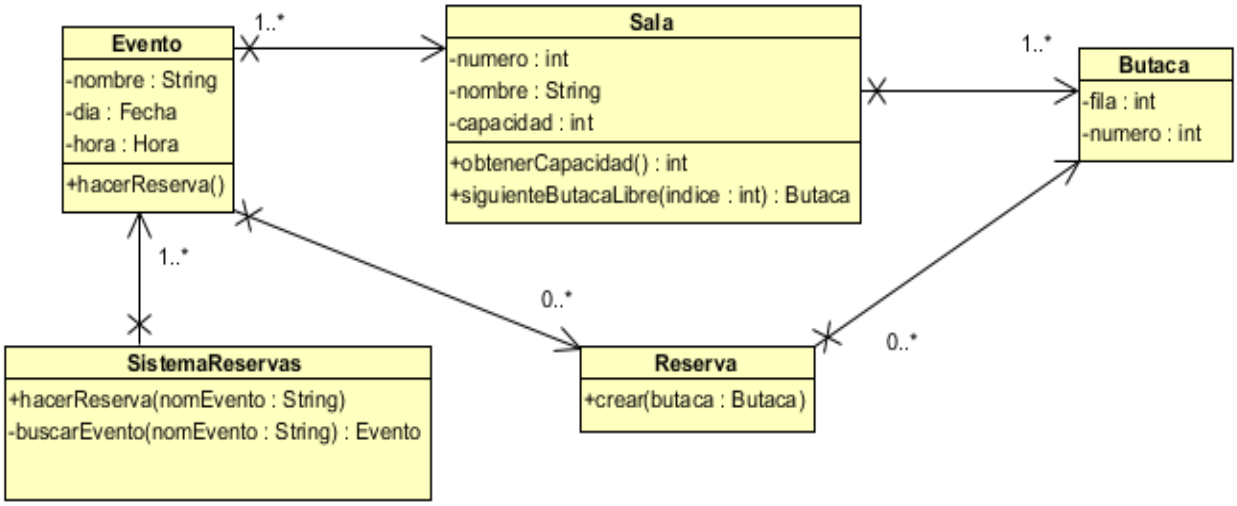


Ejercicio 2. El siguiente diagrama de clases representa hoteles con sus habitaciones y las reservas hechas por sus clientes:



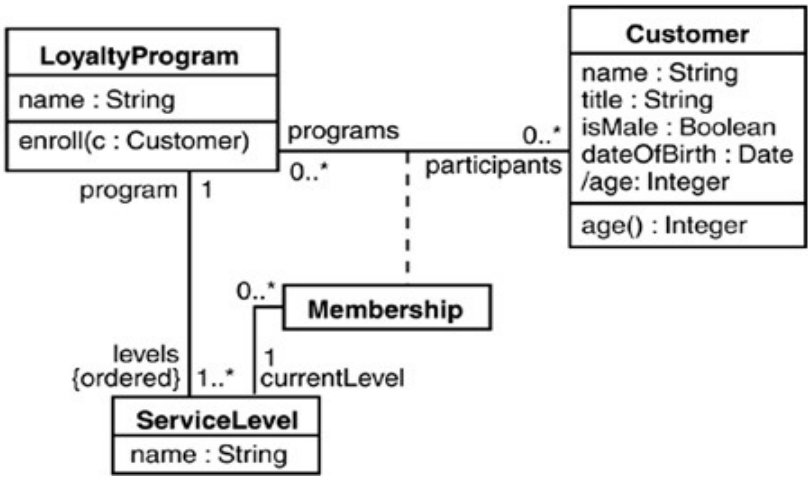
a) Complétalo con los atributos de las clases y el nombre, roles, multiplicidad y navegabilidad de las asociaciones, haciendo las suposiciones que consideres oportunas.
b) Implementa en Java el resultado obtenido.
c) Implementa en Ruby el resultado obtenido.

Ejercicio 3. A partir del siguiente diagrama de clases.



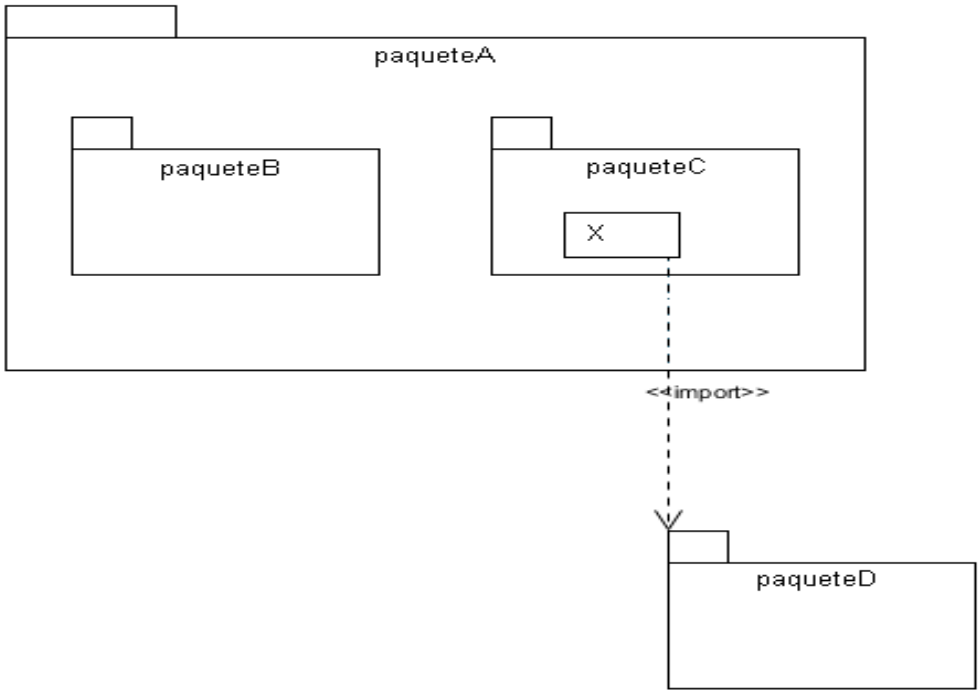
a) Indica el nombre o roles de las asociaciones.
b) ¿La asociación que existe entre Sala y Butaca podría ser una agregación?¿Y una composición?
c) Partiendo de una sala, ¿podría saberse para qué eventos está siendo usada?
d) Escribe el código Java correspondiente.
e) Escribe el código Ruby correspondiente.

Ejercicio 4. Teniendo en cuenta el siguiente diagrama de clases:



a) Escribe el código Java correspondiente, sin olvidar las asociaciones y su navegabilidad y haciendo uso del nombre de los roles que figuran en el diagrama.
b) Impleméntalo ahora en Ruby.
c) Si modificamos la navegabilidad en el siguiente sentido:
Customer ----> LoyaltyProgram ---> ServiceLevel ---> Membership
¿Qué habría que modificar en el código desarrollado anteriormente?
d) ¿Qué implicación tendrá la restricción {ordered}? ¿qué habría que hacer en el código para asegurar que se cumple?

Ejercicio 5. Dado el siguiente diagrama de paquetes:



a) Indica cómo implementarías esta estructura en Java y en Ruby

b) Justifica si sería accesible en Java un atributo de la clase X declarado con visibilidad de paquete desde:

- Una clase de paqueteB
- Una clase de paqueteA que no esté en paqueteB ni en paqueteC
- Una clase de paqueteD

Ejercicio 6. A partir del siguiente esquema de diagrama de clases (siguiente página):

a) La asociación que hay entre Empresa y Departamento, ¿es de tipo agregación o composición? ¿podría ser del otro tipo? Razona por qué.

b) ¿Cuál es la multiplicidad de dicha asociación en el lado de Empresa? ¿podría ser distinta? Razona por qué.

c) La asociación que hay entre Departamentos, ¿es de tipo agregación o composición? ¿podría ser del otro tipo? Razona por qué

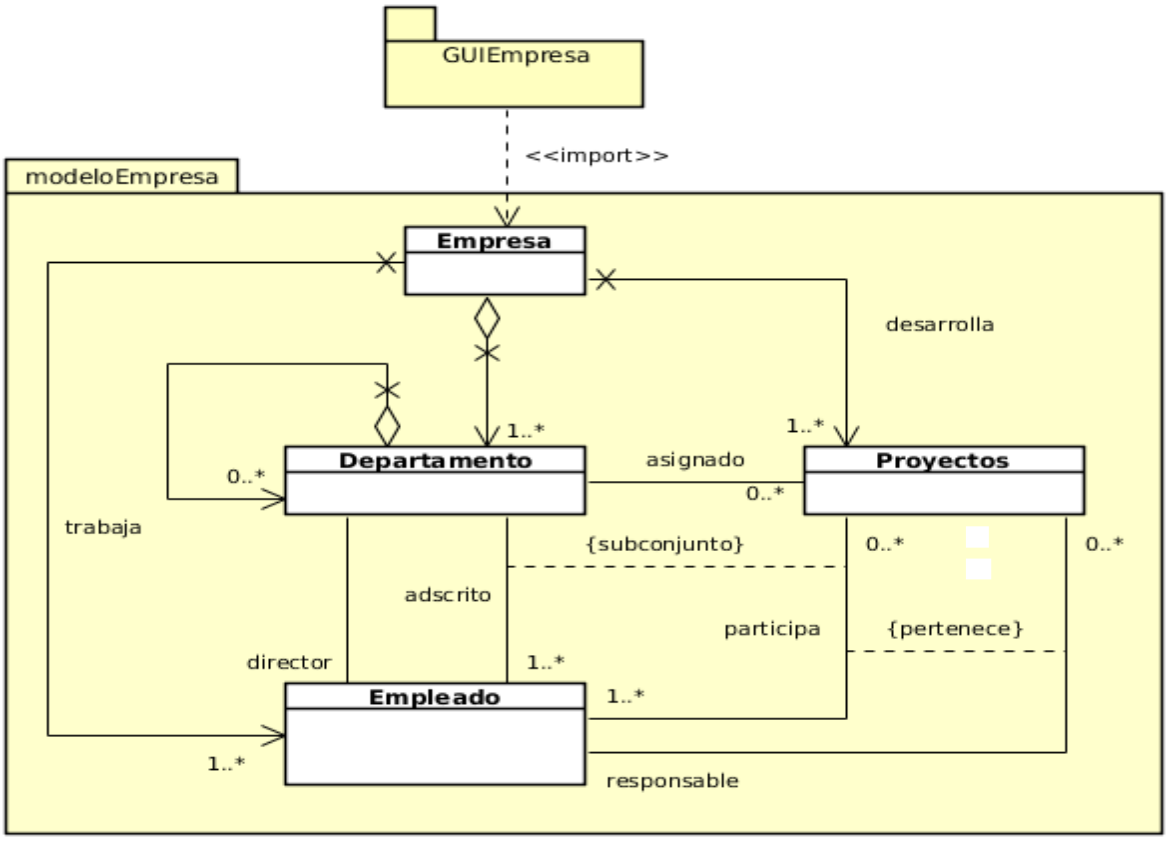
d) ¿Qué son GUIEmpresa y modeloEmpresa?

e) La relación que hay entre GUIEmpresa y Empresa ¿de qué tipo es?, es decir ¿qué significa <<import>> sobre ella?

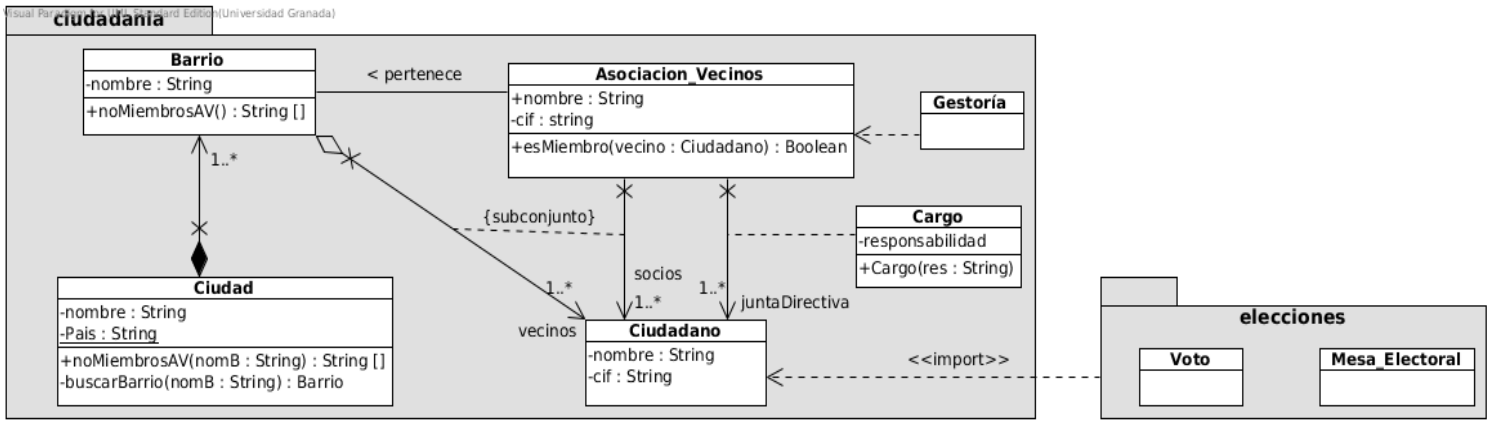
f) ¿Cuál es la navegabilidad de la asociación adscrito entre Empleado y Departamento? ¿y de la asociación desarrolla entre Empresa y Proyecto?

g) ¿Qué representa la línea discontinua entre las asociaciones adscrito y participa y cuál es su significado en este diagrama?

h) Hay dos asociaciones entre Proyecto y Empleado, ¿qué representan?



Ejercicio 7 (RESUELTO / Examen 13/14). Partiendo del siguiente diagrama de clases de UML, resolver las cuestiones planteadas a continuación



a) ¿Qué cambio habría que hacer en el diagrama de clases para modelar que un barrio puede tener varias asociaciones de vecinos?

b) ¿Desde un objeto de la clase Ciudadano puede saberse si es presidente de la asociación de vecinos de su barrio?¿Por qué?

c) ¿De qué tipo es la relación entre Ciudad y Barrio? ¿qué significa?

d) ¿De qué tipo es la relación entre Barrio y Ciudadano? ¿qué significa? ¿podría ser de otro tipo?

e) ¿Qué significa la línea discontinua con la indicación "{subconjunto}" entre la asociaciones de Barrio---Ciudadano y Asociacion_Vecinos---Ciudadano?

f) Si la junta directiva de una asociación de vecinos está formada por exactamente 6 ciudadanos, ¿cómo lo indicarías en el diagrama de clases?

g) ¿Qué significa que la clase Cargo esté ligada a la asociación entre Asociacion_Vecinos y Ciudadano?

h) Implementa en Ruby los consultores/modificadores básicos de la clase Asociación_Vecinos.

i) ¿Cómo debería ser el constructor de Ciudadano para que inicialice el estado completo de una instancia? Impleméntalo en Java y Ruby.

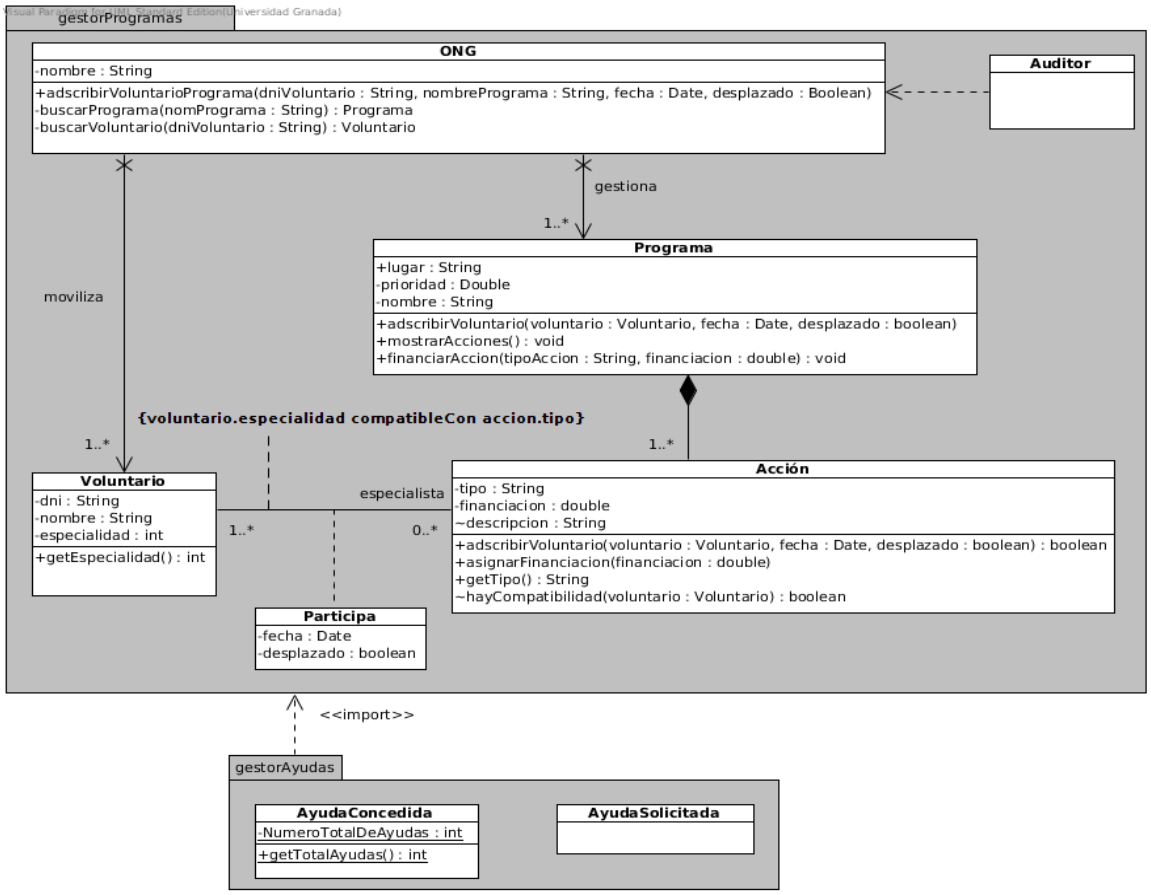
j) Suponiendo que la visibilidad de todas las clases del diagrama es pública y que estamos codificando la clase Voto en Java ¿A qué otras clases puede acceder ésta usando directamente su nombre, sin indicar el nombre del paquete al que pertenecen?

k) Define en Java los atributos de referencia de la clase Barrio.

l) ¿Desde qué clases es accesible el atributo nombre de la clase Asociacion_Vecinos?

m) Implementa en Java y Ruby las clases Asociacion_Vecinos y Mesa_Electoral completas (Ojo: no incluir nada que no esté en el diagrama de clases proporcionado).

Ejercicio 8 (Examen 12/13). Partiendo del siguiente diagrama de clases de UML, responde verdadero (V) o falso (F) a las cuestiones:



Desde la clase AyudaSolicitada se puede acceder a todos los elementos públicos del paquete GestorProgramas.	
Un voluntario puede participar en cualquier acción de un programa sin ningún tipo de restricción.	
El estado de un objeto de la clase Auditor viene determinado por el estado de un objeto de la clase ONG.	
Un voluntario podría participar en acciones de distintos programas.	
Un voluntario puede pertenecer a varias ONG.	
Cuando se define un objeto de la clase Acción, éste tiene que asociarse a un determinado objeto de la clase Programa.	
En una acción puede participar más de un voluntario como especialista.	
Desde un objeto de la clase ONG se puede llegar a conocer a todos los especialistas de una determinada acción en un programa.	
El estado de un objeto Voluntario está exclusivamente determinado por su dni, nombre y especialidad.	
Todos los métodos de la clase Acción pueden ser accedidos desde la clases AyudaConcedida.	

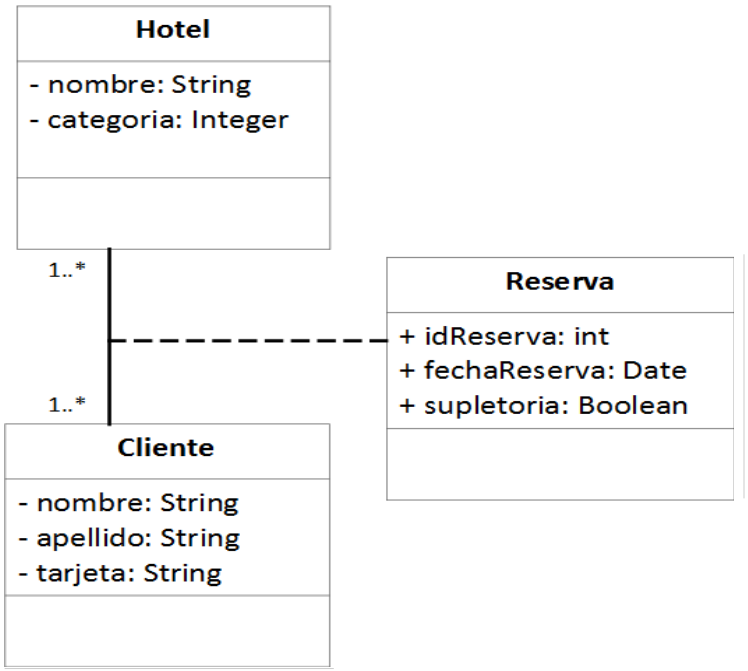
Ejercicio 9 (Resuelto / Examen 13/14). Partiendo del diagrama de clases del ejercicio 8 responde a las siguientes preguntas:

a) Usando la siguiente nomenclatura: AS = Asociación, CO = Composición, AG = Agregación, DE = Dependencia, CA = Clase Asociación y RE = Restricción, etiqueta los elementos correspondientes en el propio diagrama de clases.

b) Implementa en Java y Ruby las clases Accion y AyudaConcedida.

Ejercicio 10. Partiendo del diagrama de clases del ejercicio 8, implementa en Java y en Ruby los atributos de referencia de todas las clases.

Ejercicio 11 (RESUELTO / Ejercicio de examen 12/13). Cuando se implemente en Java el siguiente diagrama de clases, responde si las afirmaciones son verdaderas (V) o falsas (F)



Hotel tendrá dos atributos de referencia, uno relativo a las reservas y otro relativo a los clientes	
Reserva tendrá los atributos de referencia: private Hotel hotel; private Cliente cliente;	

Ejercicio 12. Modela (obtén el diagrama de clases) de los siguientes supuestos, incluyendo en el modelo todo lo que conozcas del tema:

a) En un restaurante se dispone de un Gestor de Menu encargado de elaborar los menús para cada mes, cada menú está compuesto por tres platos: un primero, un segundo y un postre. A cada plato le corresponde una receta. Los ingredientes de la receta lo componen una lista de productos y las cantidades necesarias de cada producto. Para elaborar un menú para un día concreto hay que proporcionar un primero, un segundo y un postre. También se tiene la restricción de no repetir ninguna receta a lo largo del mes.

b) En una carrera de relevos se inscriben equipos de atletas, cada equipo lo forman 4 deportistas, cada uno corre una distancia determinada tras la cual entrega el testigo al siguiente corredor del equipo. Para participar hay que inscribirse en la carrera. Una vez finalizada, reciben medalla (oro, plata y bronce) los tres equipos que primero lleguen a la meta, también se proporciona medalla (oro, plata y bronce) a los corredores que hayan obtenido los tres mejores tiempos.

c) Un alumno está matriculado en una serie de asignaturas de una determinada titulación, las asignaturas están formadas por grupos de tal forma que cada alumno asiste a clase en un determinado grupo, al comienzo del curso se le asigna a cada alumno un grupo de cada asignatura en la que esté matriculado.

d) Una cadena de hoteles posee hoteles de distintas categorías en distintas ciudades de España, los hoteles están compuesto por habitaciones y cada habitación es tipo individual, doble o suite. Los clientes de la cadena de hoteles hacen reservas para una determinada ciudad y un número de noches en una determinada fecha, siempre que haya disponibilidad en alguno de los hoteles de esa ciudad.

e) Una empresa de gestión de eventos culturales se encarga la gestión de un conjunto de salas en las que se celebran dichos eventos (teatro, conciertos...), cada sala tiene una capacidad que viene dada por el número de butacas que la compone. Para realizar la programación de eventos, se asigna a cada evento una sala y las fechas en las que se va a llevar a cabo. La empresa también se encarga de la venta de entradas de todos los eventos programados"

f) El juego de Las torres de Hanoi consiste en tres varillas verticales. En una de las varillas se apila un número indeterminado de discos que determinará la complejidad de la solución, por regla general se consideran ocho discos. Los discos se apilan sobre una varilla en tamaño decreciente. No hay dos discos iguales, y todos ellos están apilados de mayor a menor radio en una de las varillas, quedando las otras dos varillas vacantes. El juego consiste en pasar todos los discos de la varilla ocupada (es decir la que posee la torre) a una de las otras varillas vacantes. Para realizar este objetivo, es necesario seguir tres simples reglas:

   a) Sólo se puede mover un disco cada vez.
   b) Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
   c) Sólo es posible desplazar el disco de arriba de la varilla

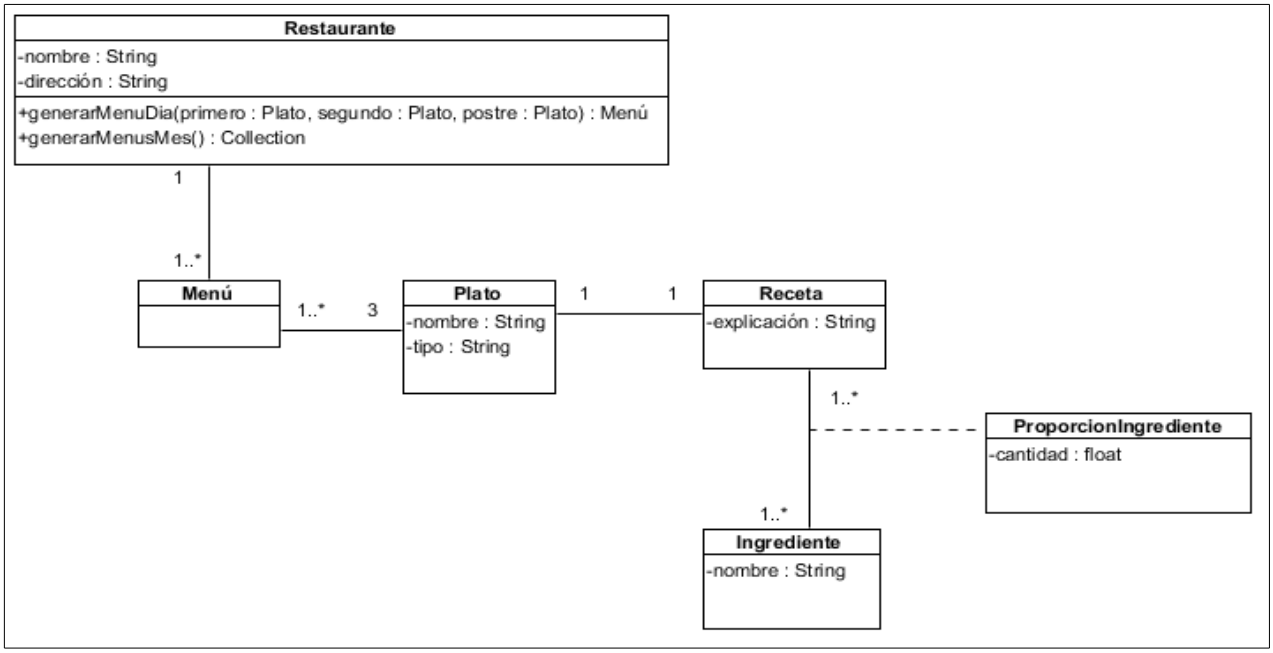
g) En geometría, un polígono regular es una figura plana compuesta por una secuencia finita de segmentos rectos consecutivos que cierran una región en el plano. Estos segmentos se denominan lados y los puntos en que se interceptan se llaman vértices. Los polígonos regulares son equiláteros (todos sus lados tienen la misma longitud) y equiángulos (poseen el mismo ángulo interior en todos sus vértices). En un polígono regular se pueden distinguir los siguientes elementos geométricos:

   - **Lado**: es cada uno de los segmentos que conforman el polígono.
   - **Vértice**: es el punto de intersección (punto de unión) de dos lados consecutivos.
   - **Diagonal**: es el segmento que une dos vértices no consecutivos.
   - **Perímetro:** es la suma de las longitudes de todos los lados del polígono.
   - **Semiperímetro**: es la mitad del perímetro.
   - **Ángulo interior**: es el ángulo formado, internamente al polígono, por dos lados consecutivos.
   - **Ángulo exterio**r: es el ángulo formado, externamente al polígono, por un lado y la prolongación de un lado consecutivo.
   - **Centro**: es el punto equidistante de todos los vértices y lados.
   - **Apotema**: es el segmento que une el centro del polígono con el centro de un lado; es perpendicular a dicho lado.
   - **Diagonales totales**: N_d = (n(n-3))/2, en un polígono de n lados.

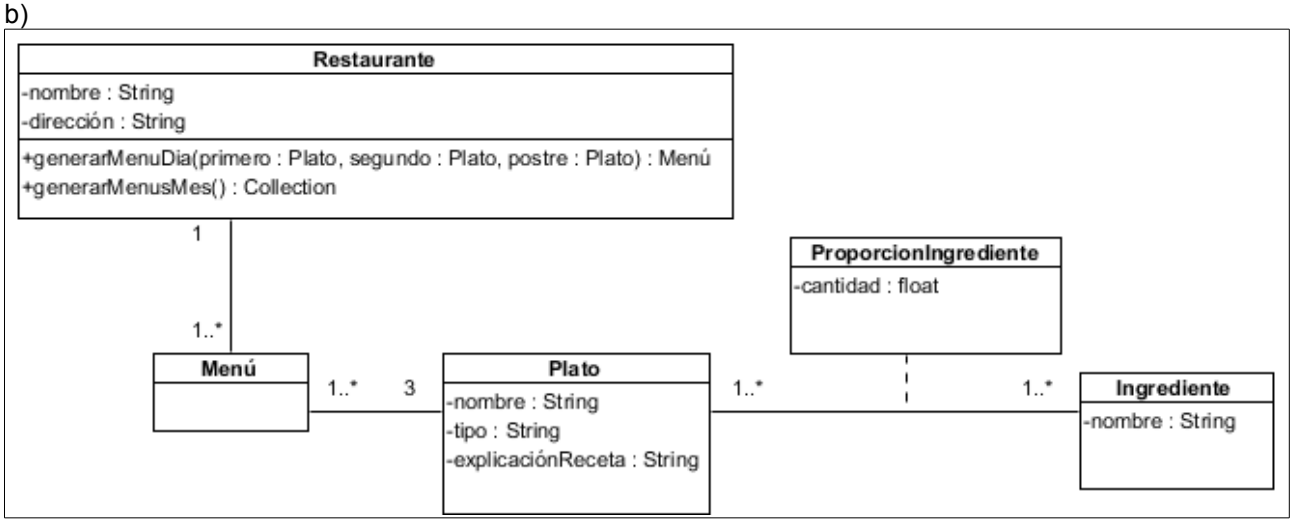
- **Intersecciones de diagonales**: N_I = (n(n-1)(n-2)(n-3))/24, en un polígono de n vértices

Ejercicio 13. A continuación se muestran varios diseños, todos congruentes con la especificación del ejercicio 12.a. Explica en qué se diferencian en cuanto al significado de los objetos de cada clase y sus relaciones y cómo afectaría a su posterior codificación. Incluye tu propia solución al ejercicio 12.a en la comparativa,

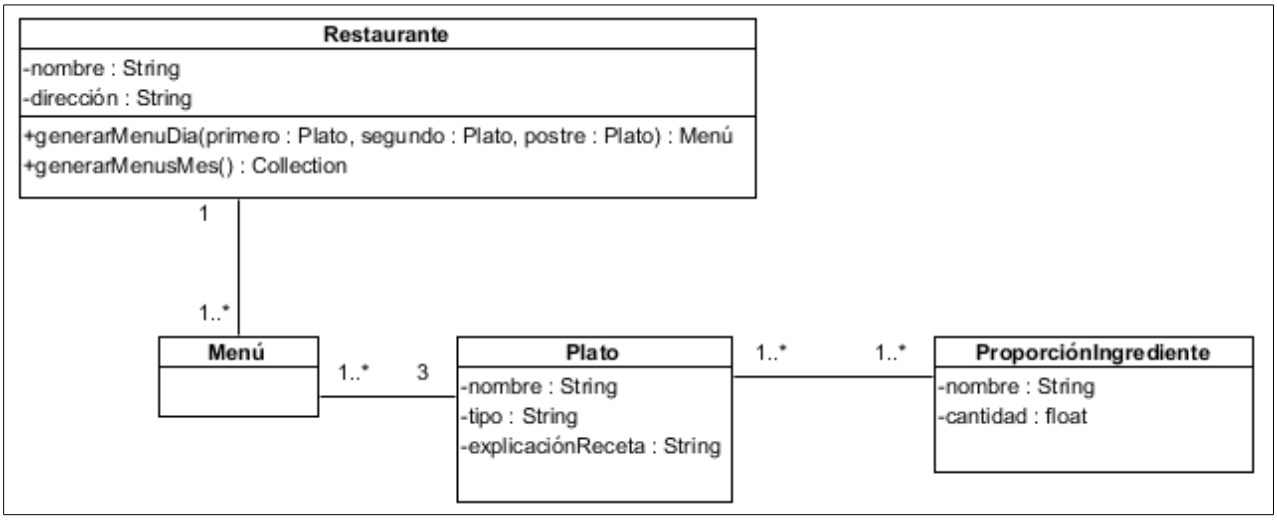
a)



b)



c)

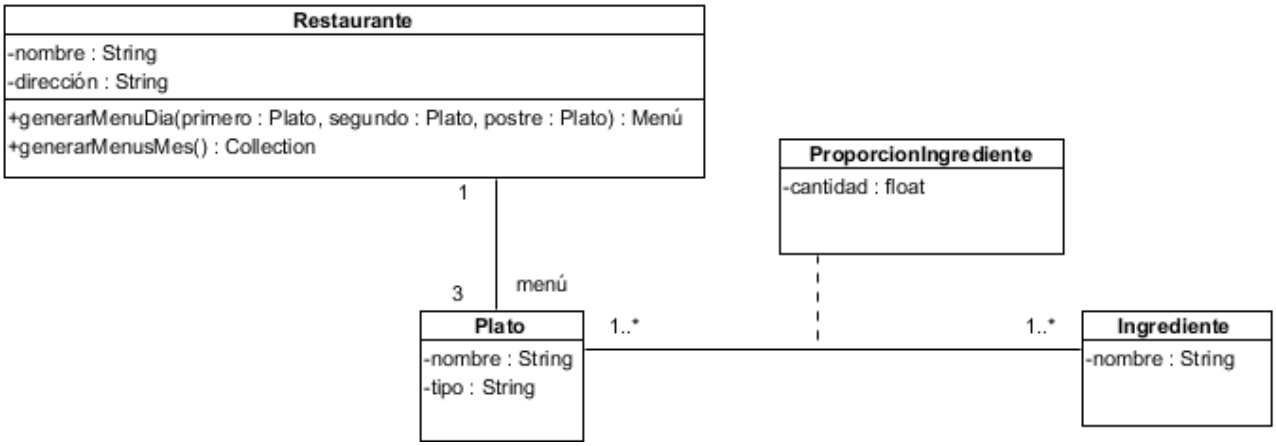


Ejercicio 14. ¿Cómo mejorarías los diseños expuestos en el ejercicio 13 usando de mejor forma la navegabilidad? Indica las navegabilidades que elegirías y explica por qué.

Ejercicio 15. ¿Cómo indicarías en el diagrama de clases que el "tipo" de un Plato puede ser únicamente "primero", "segundo" o "postre"?

Ejercicio 16. Incluye en uno de los diagramas del ejercicio 13 las operaciones y atributos que consideres necesarios para el caso en que estuviésemos interesados en calcular las calorías de los menús.

Ejercicio 17. ¿El siguiente diseño es correcto según la especificación del ejercicio 12.a? ¿por qué?



Algunos ejercicios resueltos

Ejercicio 7

a) En la relación que une Barrio con Asociación_Vecinos, junto a Asociación_Vecinos debe aparecer la cardinalidad 1..*

b) No, la relación no es navegable desde Ciudadano hasta Asociación_Vecinos.

c) Es una asociación de composición, relación fuerte. Las partes no tienen sentido sin el todo. Significa que no puede haber Barrios sin que formen parte de una Ciudad

d) Es una asociación de agregación, relación débil. Significa que los ciudadanos son parte del barrio. Podría haber barrios sin ciudadanos (barrios nuevos sin habitar, por ejemplo). Podría haber ciudadanos que no fueran vecinos de un barrio, que fueran por ejemplo de un poblado, donde Poblado fuera otra entidad diferente de Ciudad.

e) Significa que todos los ciudadanos socios de una asociación deben ser vecinos del barrio, es decir, un subconjunto de los vecinos de un barrio son socios de la asociación de vecinos.

f) En la línea que une asociación de vecinos con ciudadano y con la palabra juntaDirectiva, se sustituye 1..* por 6 para indicar la cardinalidad de la relación.

g) Es una clase asociación. Se utiliza porque la asociación tiene atributos propios (la responsabilidad) y complementa la asociación indicando el cargo que tiene un ciudadano concreto en la juntaDirectiva de su asociación de vecinos.

h) attr_accessor :nombre, :cif

i) **Ruby**

```
def initialize(nom,cif)
        @nombre=nom
        @cif=cif
end
```

Java

```
Ciudadano(String nom, String cif){
        this.nombre=nom;
        this.cif=cif;
}
```

j) Puede acceder a Ciudadano (porque lo importa) y a Mesa_Electoral (porque está en su mismo paquete)

k) cargo1 == cargo2; Falso, referencian a distintos objetos

cargo1.equals(cargo2); Verdadero, tienen el mismo estado

cargo1 == cargo3; Verdadero, referencian al mismo objeto

cargo1.equals(cargo3); Verdadero, tienen el mismo estado

l) private ArrayList <Ciudadano> vecinos;

private Asociacion_Vecinos asociacion;

nombre es un atributo básico, no de referencia.

m) El atributo nombre es público, es decir, es potencialmente accesible desde cualquier clase de su paquete o de otros paquetes.

n) **Java**

```
package ciudadania;

class Asociacion_Vecinos{
        public String nombre;
        private String cif;
        private Barrio barrio;
        private ArrayList <Ciudadano> socios;
        private ArrayList<Cargo> cargos;
        public Boolean esMiembro(Ciudadano vecino) { }
}
package elecciones;
import ciudadania.Ciudadano;
class Mesa_Electoral { }
```

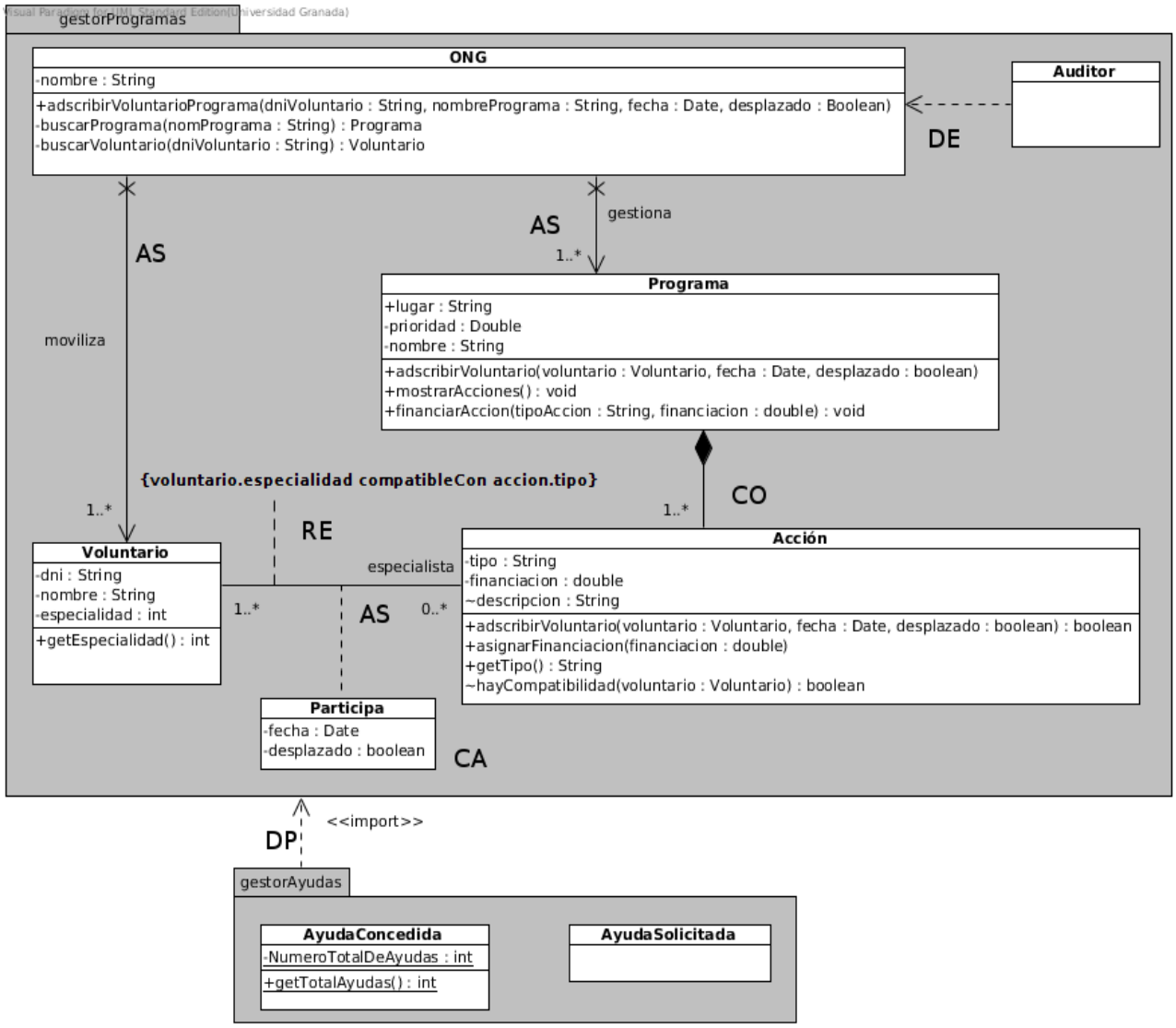
Ruby

```
module ciudadania

class Asociacion_Vecinos
        def initialize (nombre,cif,barrio)
                @ nombre = nombre
                @ cif = cif
                @barrio = barrio
                @socios = Array.new
                @cargos = Array.new
        end
        def esMiembro(vecino)
        end
end
module elecciones
require_relative 'ciudadania.Ciudadano'
class Mesa_Electoral
end
```

Ejercicio 9

a)



b)

Java: Accion

```
package gestorProgramas;
import java.util.ArrayList;
import java.util.Date;

public class Accion {
    public String tipo;
    private double financiacion;
    String descripcion;
    private ArrayList<Participa> especialistas = new ArrayList();
    private Programa miPrograma;
```

```
    public boolean adscribirVoluntario(Voluntario voluntario, Date
                                       fecha, boolean          desplazado)
        {return false;}
    public void asignarFinanciacion(double financiacion){}
    public String getTipo(){return tipo;}
    boolean hayCompatibilidad(Voluntario voluntario){return false;}
}
```

Java: AyudaConcedida

```
package gestorAyudas;
import gestorProgramas.*;
public class AyudaConcedida {
   private static int NumeroTotalDeAyudas;
   public static int getTotalAyudas(){return NumeroTotalDeAyudas;}
}
```

Ruby : Accion

```
# definida en el archivo gestorProgramas.rb
class Accion
        def initialize(tipo, financiacion,descripcion,programa)
                @tipo = tipo,
                @financiacion = financiacion
                @descripcion = descripcion
                @especialistas =Array.new
                @miPrograma = programa
        end
        att_reader :tipo

         def adscribirVoluntario(voluntario,fecha,desplazado)
         end
         def asignarFinanciacion(financiacion)
         end
         def hayCompatibilidad(voluntario)
         end
end
```

Ruby: AyudaConcedida

```
require_relative 'gestorProgramas'
class AyudaConcedida
        @@NumeroTotalDeAyudas
        def self.getTotalAyudas
        end
end
```

Ejercicio 11

Hotel tendrá dos atributos de referencia, uno relativo a las reservas y otro relativo a los clientes	Falso. Hotel tendrá únicamente un atributo relativo a las Reservas, en concreto una colección que almacene las reservas realizadas por los clientes.
Reserva tendrá los atributos de referencia: private Hotel hotel; private Cliente cliente;	Verdadero. Cada reserva en particular está asociada a un único hotel y cliente.

Tema 2: Lección 3

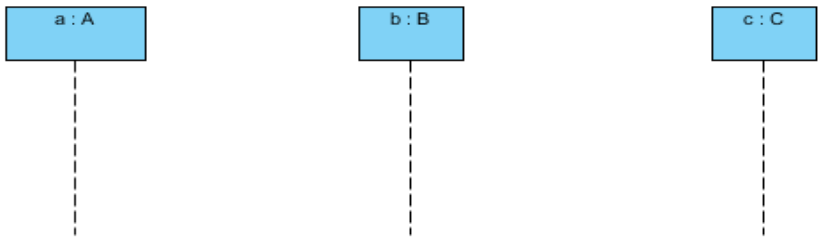
Ejercicio 1. En general, ¿qué representan los diagramas de interacción de UML? ¿Cuáles son sus componentes principales?

Ejercicio 2. Establece la correspondencia entre los diagramas de secuencia y los diagramas de comunicación, indicando cómo se representan en cada uno los siguientes elementos: objetos, mensajes, canal de comunicación, estructuras de control, subordinación en el envío de mensaje y orden de un mensaje.

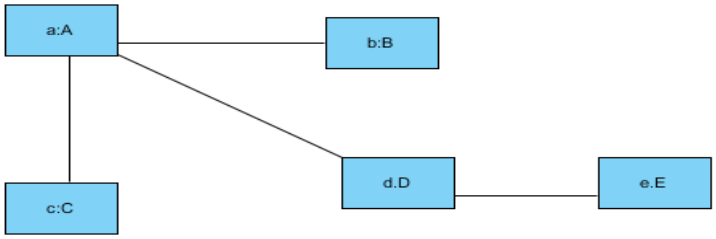
Ejercicio 3. ¿Qué relación existe entre el diagrama de clases y los diagramas de interacción?

Ejercicio 4. A partir del siguiente esquema, elabora un diagrama de secuencia genérico en el que haya al menos dos fragmentos combinados distintos.

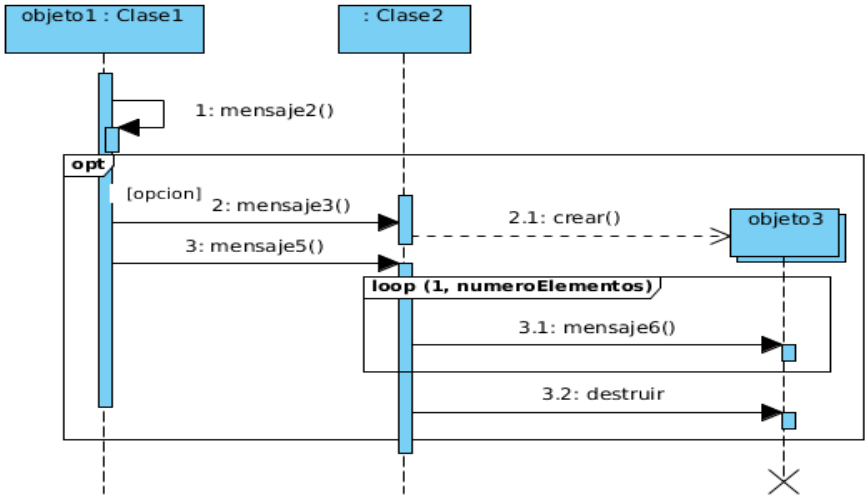
Tradúcelo a su correspondiente diagrama de comunicación e impleméntalo en Java y Ruby.



Ejercicio 5. A partir del siguiente esquema elabora un diagrama de comunicación genérico en el que haya al menos una estructura de control iterativa y otra selectiva. Tradúcelo a su correspondiente diagrama de secuencia e impleméntalo en Java y Ruby.



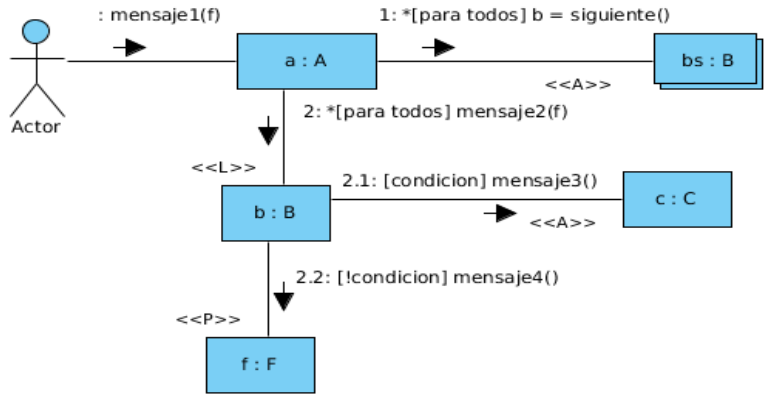
Ejercicio 6: Dado el siguiente Diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones



El envío de mensaje 1 (**mensaje2()**) es un envío de mensaje a **self** y además **recursivo.**	
El objeto **objeto3** es un objeto que vive sólo en esta interacción.	
En la clase **Clase2** tienen que estar definidos los siguientes métodos: mensaje3(), mensaje5() y mensaje6().	
El **fragmento combinado** tipo **loop** solo se puede usar para el envío de mensajes a multiobjetos, tal y como está representado en el ejemplo.	
La representación del multiobjeto está mal, falta especificar la clase a la que pertenecen los objetos que forman ese multiobjeto.	
La numeración está mal, los envíos de mensaje 3.1 y 3.2 deberían ser 2.2 y 2.3	

El siguiente código Ruby **no es correcto**:
class Clase 2
  def mensaje5
  objeto3.each [|obj| obj.mensaje6()]
  end
end	

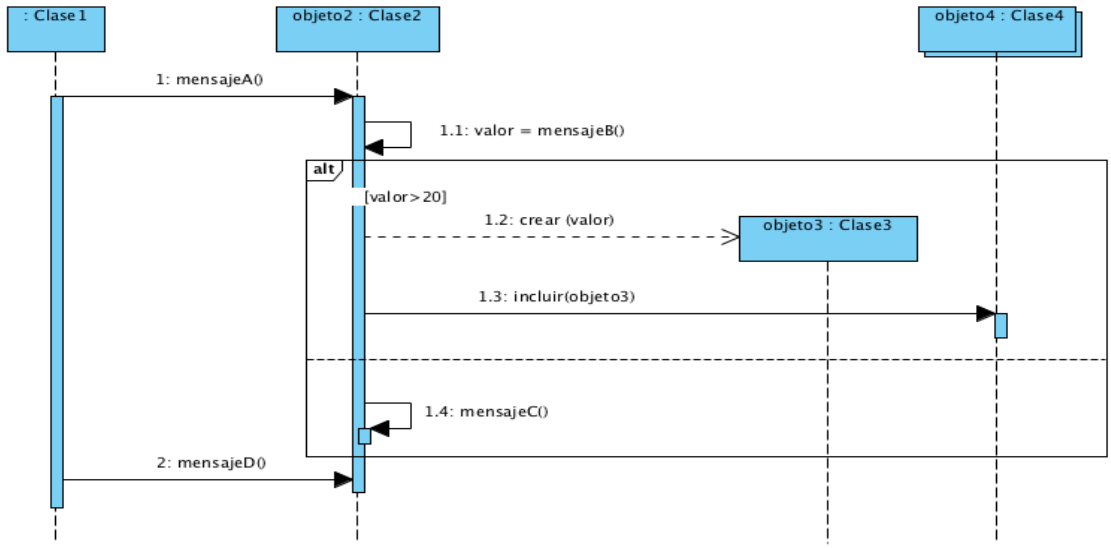
Ejercicio 7: Dado el siguiente Diagrama de comunicación, responde verdadero (V) o falso (F) a las siguientes cuestiones



El enlace o canal de comunicación estereotipado como **<<P>>** no puede ser de ese tipo ya que el objeto **f** no ha entrado como parámetro a la operación.
La estructura de control usada en los envíos de mensajes números **2.1** y **2.2** es la estructura **if**(condicion){...} **else** {...}
La implementacion de **siguiente()** debe estar en la clase **B** y en ella se envían los mensajes mensaje3() y mensaje4() a los objetos **c** y **f** respectivamente.
A los envíos de mensaje 2.1 y 2.2 les falta *[para todos]

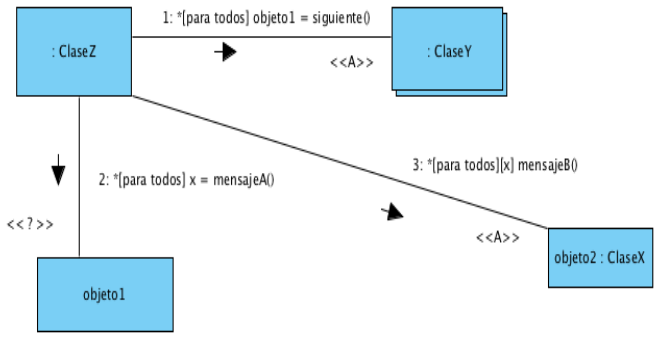
El siguiente código Java **es correcto**:
public class A {
public void mensaje1(F f){
for(B b:bs){
b.mensaje2(f);
}
}
}

Ejercicio 8. A partir del siguiente diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones



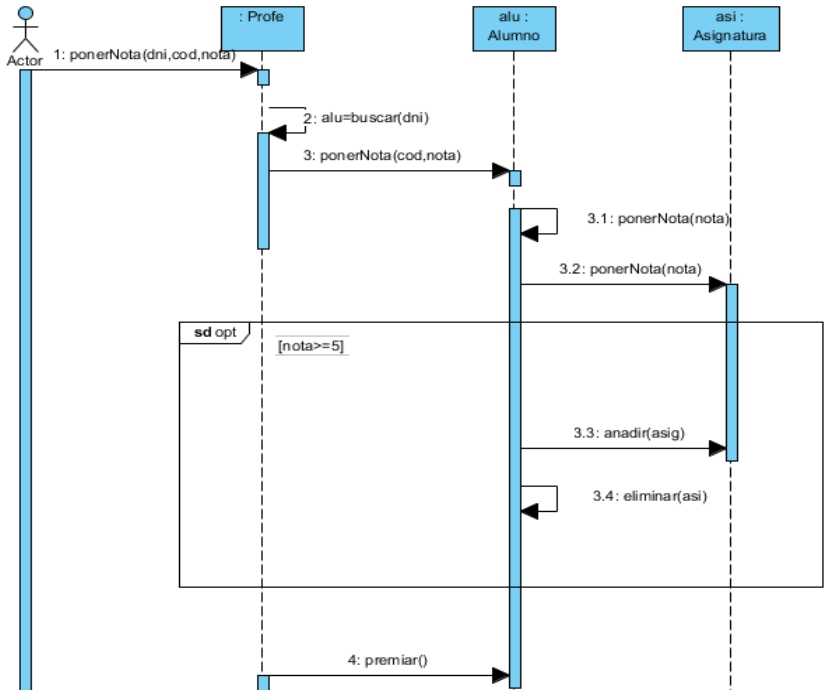
La clase **Clase2** debe tener un método que se llame **mensajeB**.
El paso numerado con un 2 debería ser 1.5.
En el diagrama de comunicación equivalente, el tipo de enlace entre el **objeto2** y el **objeto3** sería **<<A>>**.
El paso 1.2 corresponde a una llamada al constructor por defecto de la clase **Clase3**.
El objeto **objeto4** es de la clase **Clase4**.
mensajeB y **mensajeC** se ejecutan de forma recursiva.
Los pasos 1.2, 1.3 y 1.4 se ejecutan cuando valor es mayor de 20.

Ejercicio 9. A parir del siguiente diagrama de comunicación, responde verdadero (V) o falso (F) a las siguientes cuestiones



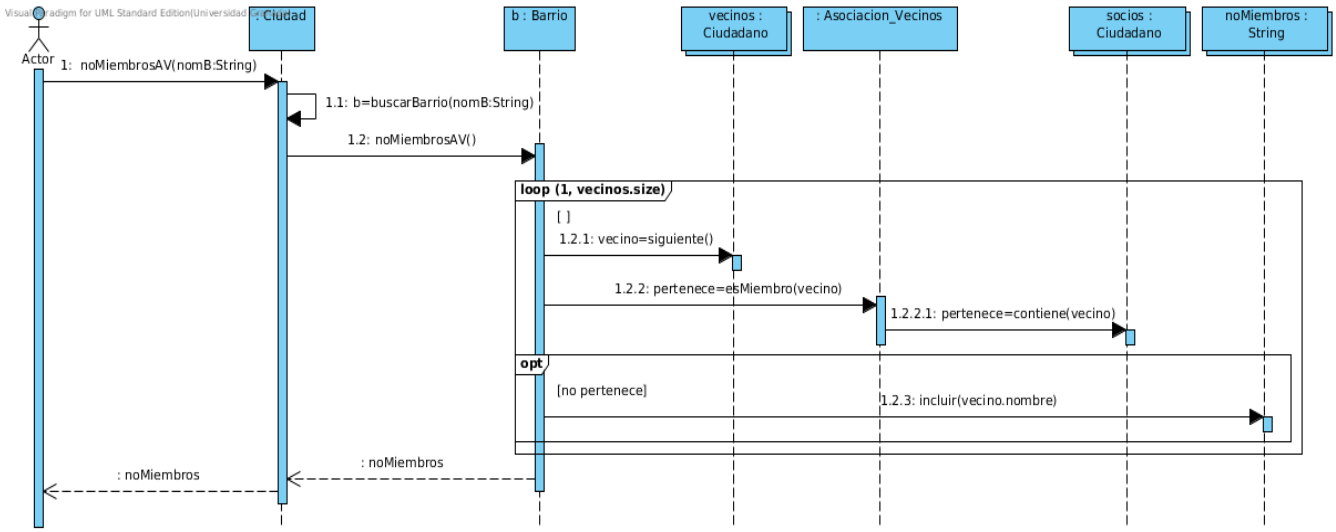
El tipo de enlace que aparece con interrogación no puede ser **<<A>>**.	
Al codificar el diagrama, las instrucciones correspondientes a los pasos 1, 2 y 3 estarán todas dentro de un mismo bucle for.	
Aunque no se indique explícitamente, es posible conocer la clase de **objeto1** por la información contenida en el diagrama.	

Ejercicio 10. A partir del siguiente diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones



El mensaje 3.1 indica que **ponerNota(cod,Nota)** debe hacerse recursivamente
La barra de activación del objeto **:Profe** está bien, ya que son envíos de mensaje asíncronos
La barra de activación del objeto **asi:Asignatura** está bien, ya que son envíos de mensajes síncronos
Los envíos de mensaje 3.1 y 3.2 no pueden tener el mismo nombre y los mismo parámetros.
El envío del mensaje 2 es un envío de mensaje a **self o this**
El operador del fragmento combinado está bien, se corresponde con una estructura de tipo **sd opt**

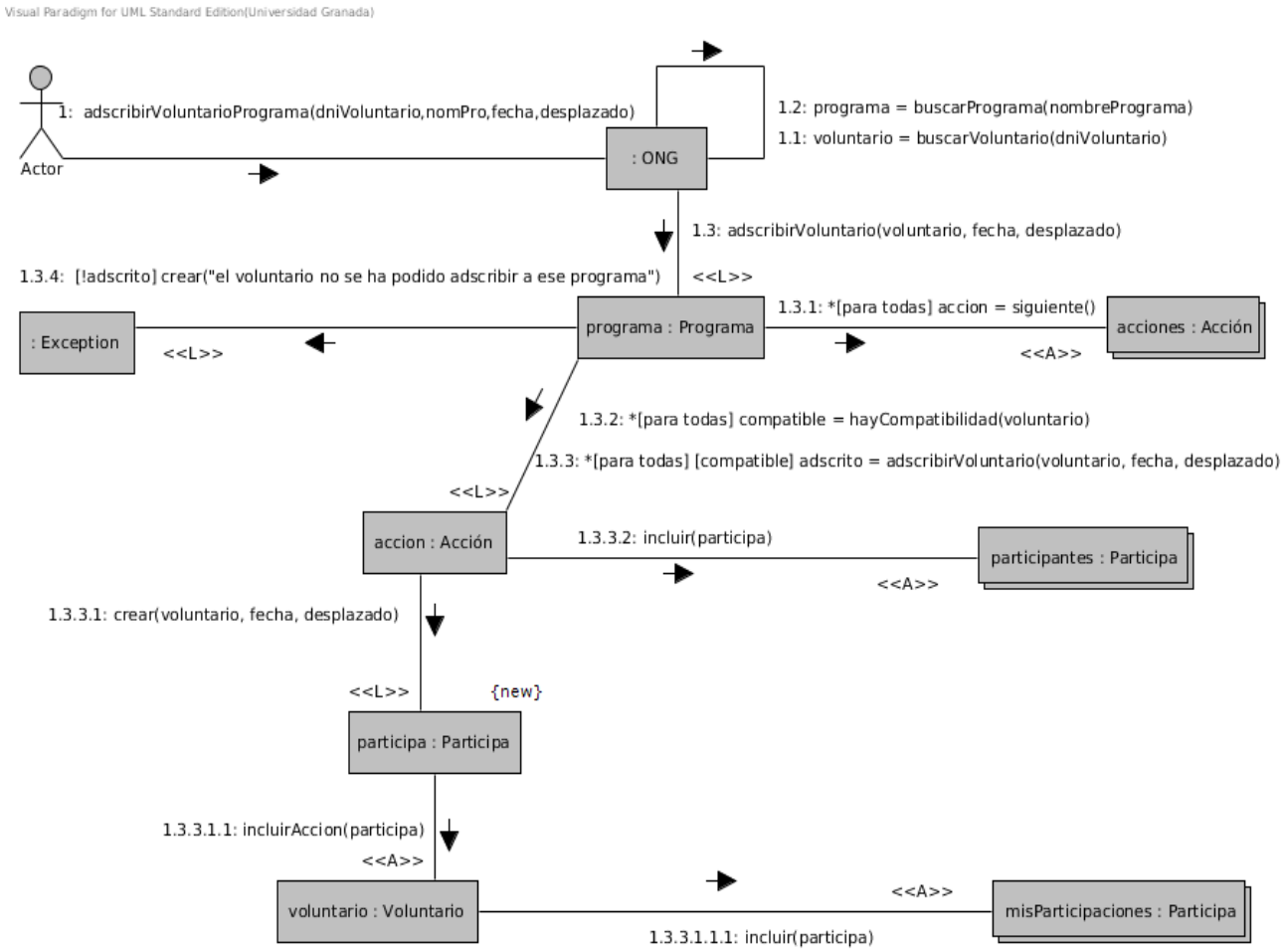
Ejercicio 11. A partir del siguiente diagrama de secuencia:



A) Tradúcelo a Diagrama de comunicación

B) Implementa en Java y en Ruby el método **noMiembrosAV()** de la clase **Barrio**

Ejercicio 12. A partir del siguiente Diagrama de comunicación:



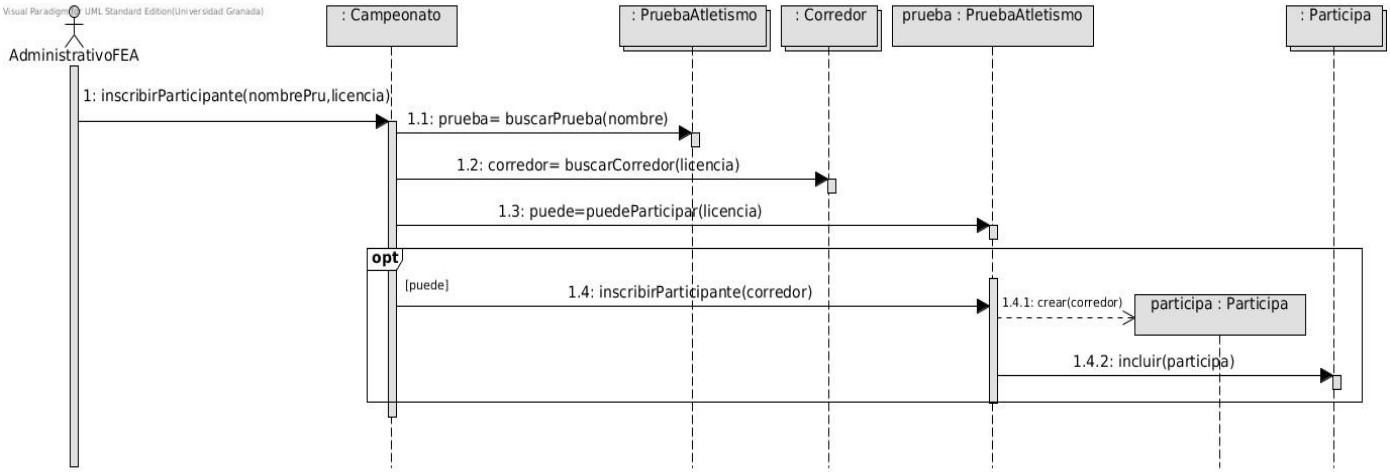
A) Responde V(verdadero) o F(falso) a las siguientes cuestiones:

En el envío de mensaje 1.2 el objeto receptor es **self/this**.	
En envío de mensaje 1.3.1 significa que a todas las acciones del programa le vamos a adscribir un voluntario.	
En el método **crear** de la clase **Participa** (1.3.3.1) se construye un enlace entre el objeto **participa** y el objeto **voluntario**.	
El enlace entre el objeto **accion** y el multiobjeto de la clase **Participa** estereotipado como **<<A>>** significa que el objeto **acción** conoce al muiltiobjeto solo para esta operación.	
El multiobjeto **misParticipaciones** enlazado con **voluntario** es un subconjunto del multiobjeto **participantes** enlazado con **accion**.	
El envío de mensaje 1.3.3 se lleva a cabo sólo si **adscrito** es verdadero.	

B) Implementa en Java y en Ruby el método **adscribirVoluntario**(...) de la clase Programa.

C) Obtén el Diagrama de secuencia de la operación **adscribirVoluntario**(...) de la clase **Accion**, incluyendo todos los envíos de mensaje subordinados al 1.3.3.

Ejercicio 13. A partir del siguiente diagrama de secuencia



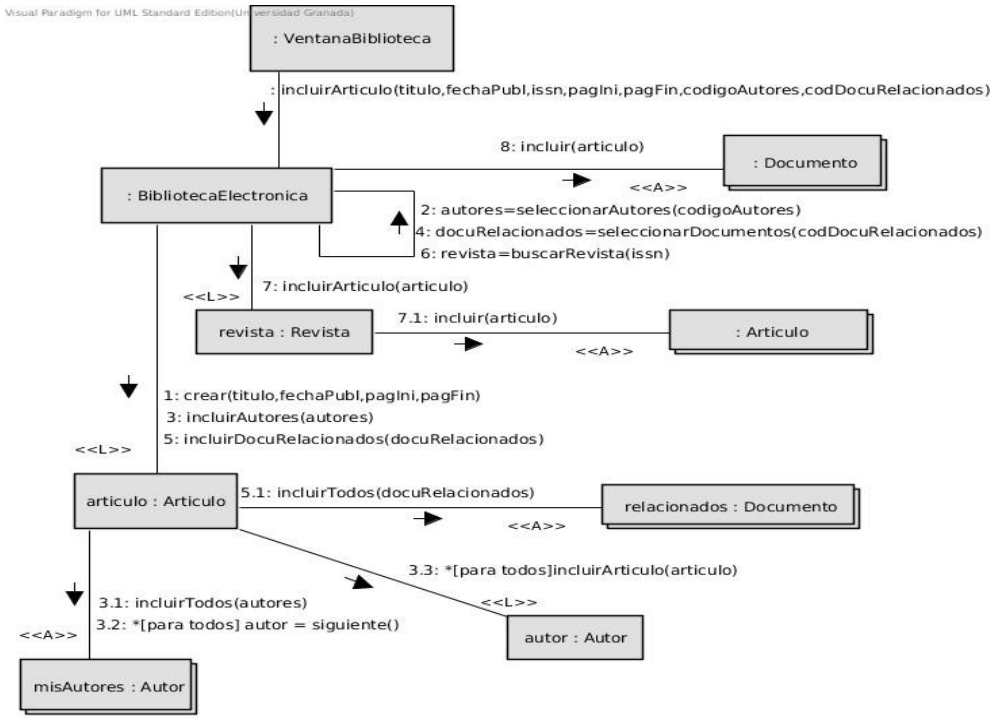
A) Responde V (verdadero) o F (Falso) a las siguientes cuestiones:

La condición del fragmento combinado **opt** está mal, tendría que estar negada	
En el envío de mensaje 1.3 tendría que entrar como parámetro el objeto **corredor** y no su **licencia**	
El envío de mensaje 1.4.2 está mal, el objeto receptor no es un multiobjeto	
El objeto **participa**:**Participa** se crea si **(puede == true)**	
El argumento del envío de mensaje 1.1 está mal debe ser **nombrePru**	
El envío de mensaje 1.4.1 se corresponde con la instanciación de un objeto de la clase **Participa**	
El objeto receptor del envío de mensaje 1.3 es **corredor**	
Los envíos de mensaje 1.4.1 y 1.4.2 deberían estar fuera del fragmento combinado **opt**	

B) Obtén el diagrama de comunicación equivalente

C) Implementa en Java y el Ruby el método **inscribirParticipante** en la clase **Campeonato**

Ejercicio 14. A partir del siguiente diagrama de comunicación



A) Responde V(verdadero) y F (falso) a las siguientes cuestiones:

Los estereotipos de visibilidad no están especificados en el diagrama	
El objeto **relacionados:Documento** es un objeto dela clase **Documento**	
El envío de mensaje 5.1 está mal numerado, debería ser 3.4	
En este diagrama se muestra que el objeto **revista:Revista** conoce a un multiobjeto de objetos de la clase **Artículo**	
La implementación del envío de mensaje 5.1 es: reacionados.incluirTodos(docusRelacionados)	

La implementación del método **incluirAutores(autores)** en la clase **Articulo** es la siguiente y es correcta:
class Articulo
private ArrayList<Autor> autores = new ArrayList();
void incluirAutores(Autor autores){
autores.addAll(autores);
for(Autor autor:autores)
autor.incluirArticulo(articulo);
}
}	

Los envíos de mensaje 2, 4 y 6 son envíos de mensajes a **self**	

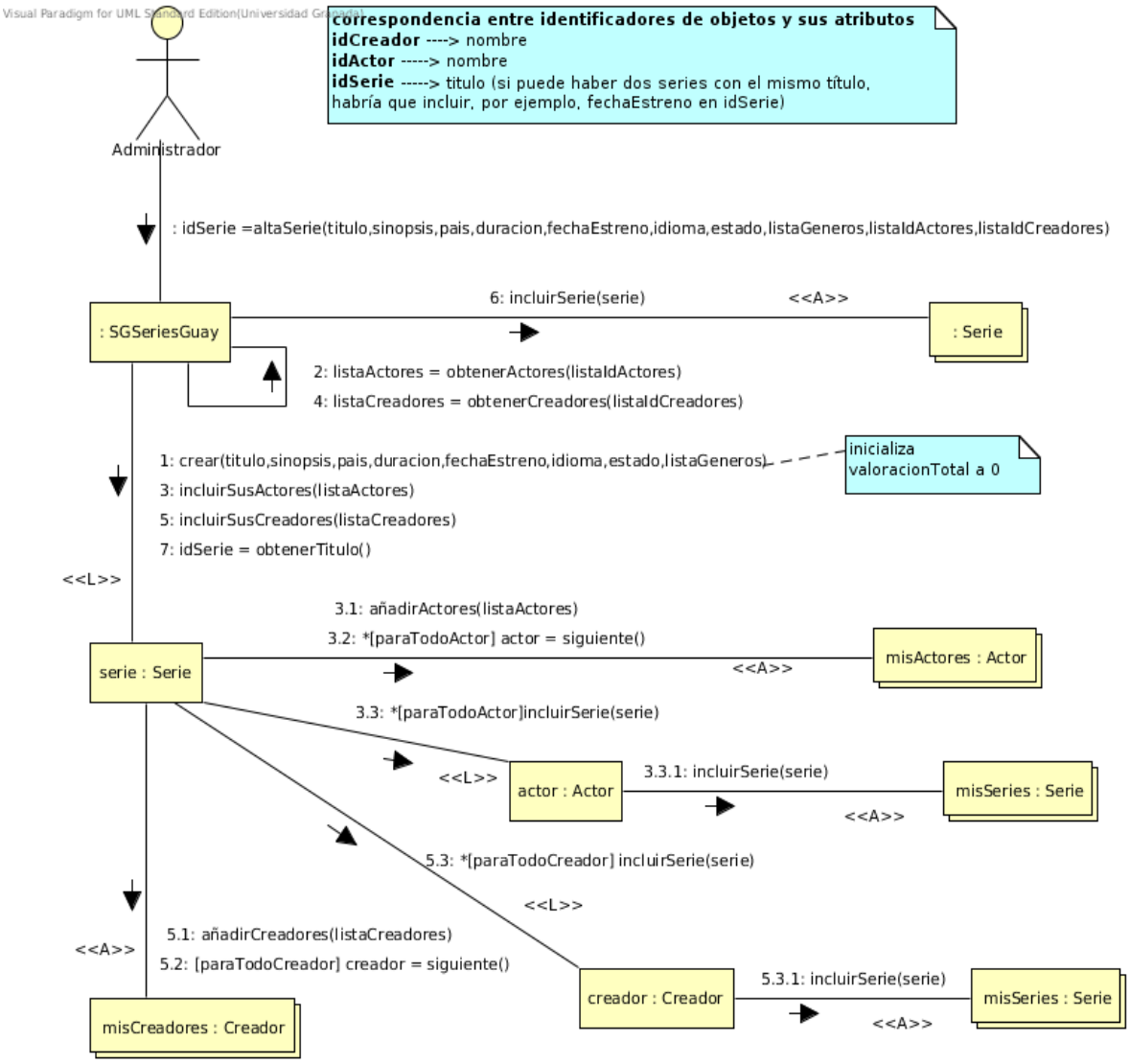
B) Impleméntalo en java y el Ruby

C) Obtén el Diagrama de secuencia de la operación **3:incluirAutores(autores)**

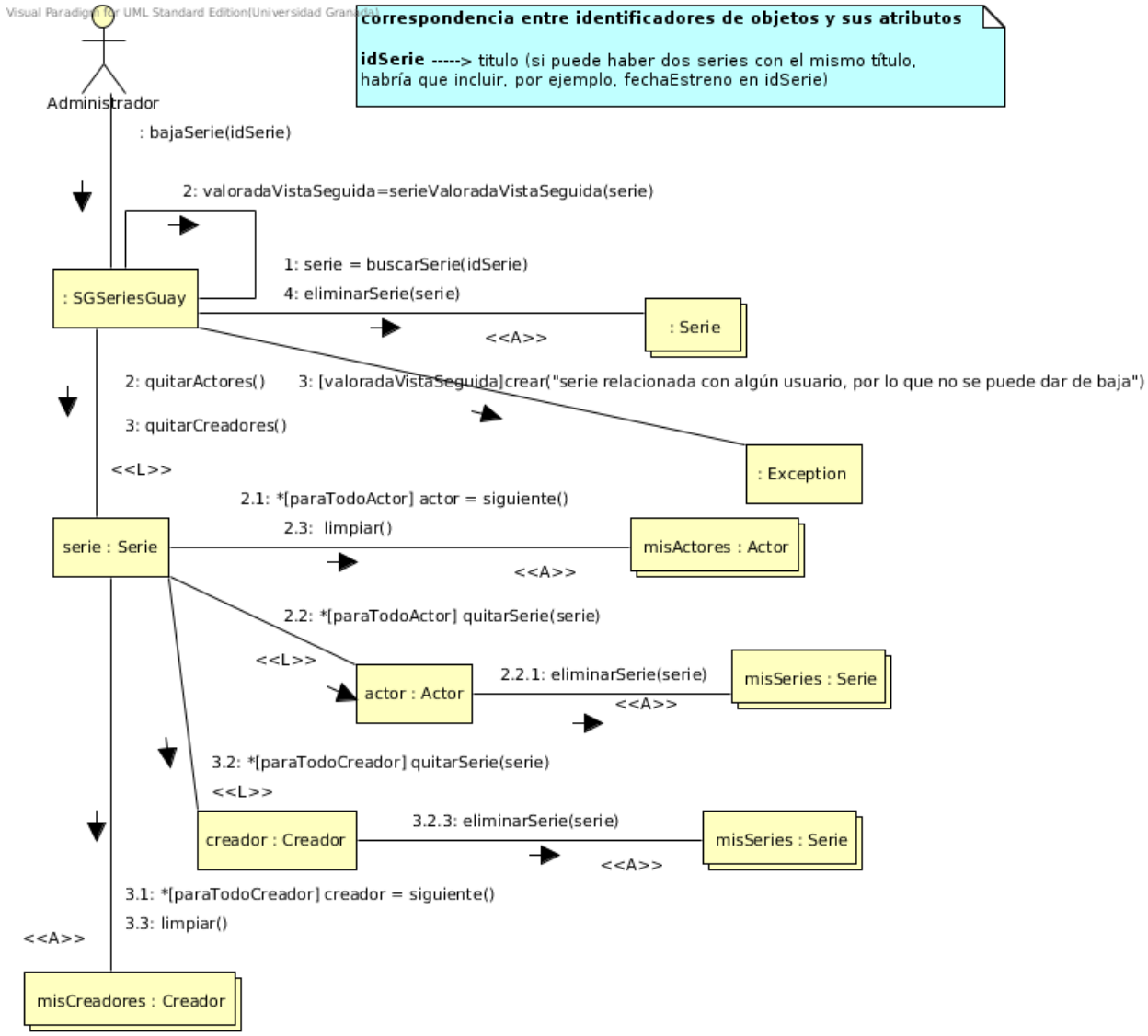
Ejercicio 15 .Traduce los diagramas de la práctica 3 de secuencia a comunicación o de comunicación a secuencia según corresponda.

Ejercicio 16. Implementa en Java y Ruby los diagramas de comunicación que se proporcionan a continuación. Traduce al menos un diagrama de comunicación de los proporcionados a su correspondiente diagrama de secuencia.

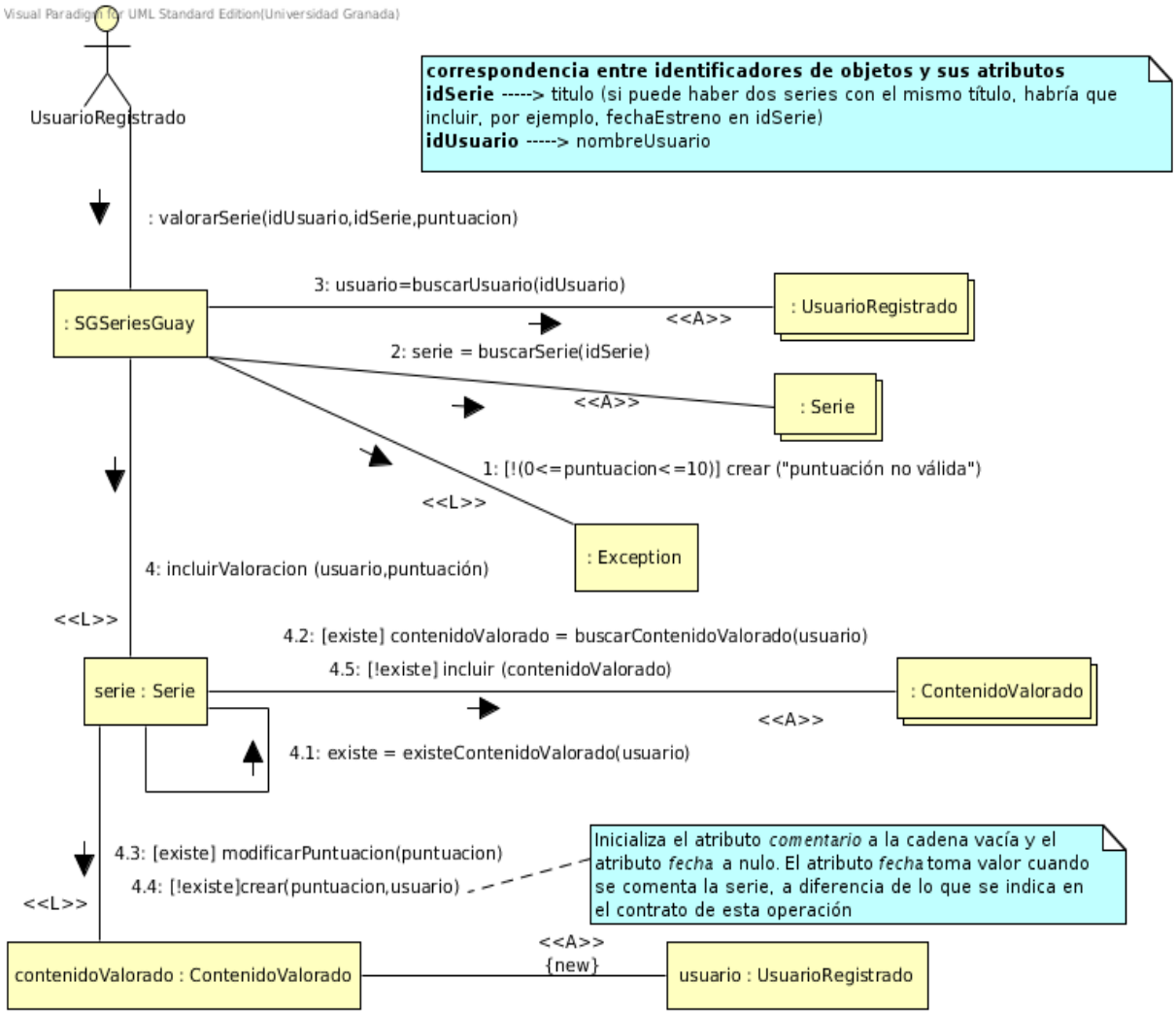
A) Incluir una nueva serie de TV en el sistema **:SGSeriesGuay**


Visual Paradigm for UML Standard Edition(Universidad Granada)
correspondencia entre identificadores de objetos y sus atributos
idCreador ----> nombre
idActor -----> nombre
idSerie -----> titulo (si puede haber dos series con el mismo título, habría que incluir, por ejemplo, fechaEstreno en idSerie)
Administrador
: idSerie =altaSerie(titulo,sinopsis,pais,duracion,fechaEstreno,idioma,estado,listaGeneros,listaIdActores,listaIdCreadores)
: SGSeriesGuay
6: incluirSerie(serie)
<<A>>
: Serie
2: listaActores = obtenerActores(listaIdActores)
4: listaCreadores = obtenerCreadores(listaIdCreadores)
1: crear(titulo,sinopsis,pais,duracion,fechaEstreno,idioma,estado,listaGeneros)
inicializa valoracionTotal a 0
3: incluirSusActores(listaActores)
5: incluirSusCreadores(listaCreadores)
7: idSerie = obtenerTitulo()
<<L>>
3.1: añadirActores(listaActores)
3.2: *[paraTodoActor] actor = siguiente()
serie : Serie
<<A>>
misActores : Actor
3.3: *[paraTodoActor]incluirSerie(serie)
<<L>>
actor : Actor
3.3.1: incluirSerie(serie)
<<A>>
misSeries : Serie
5.3: *[paraTodoCreador] incluirSerie(serie)
<<L>>
<<A>>
5.1: añadirCreadores(listaCreadores)
5.2: [paraTodoCreador] creador = siguiente()
misCreadores : Creador
creador : Creador
5.3.1: incluirSerie(serie)
<<A>>
misSeries : Serie

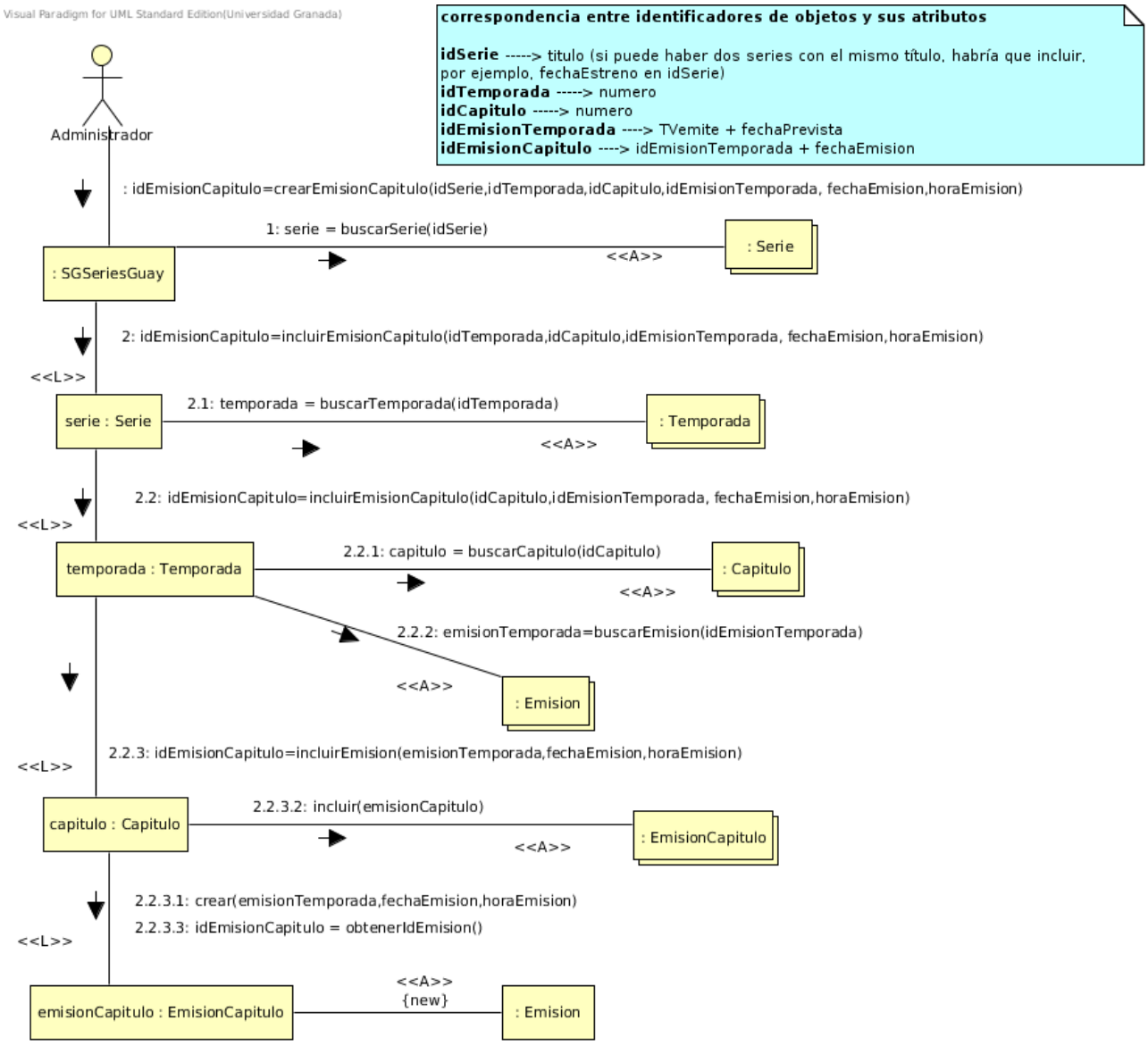
B) Eliminar una serie



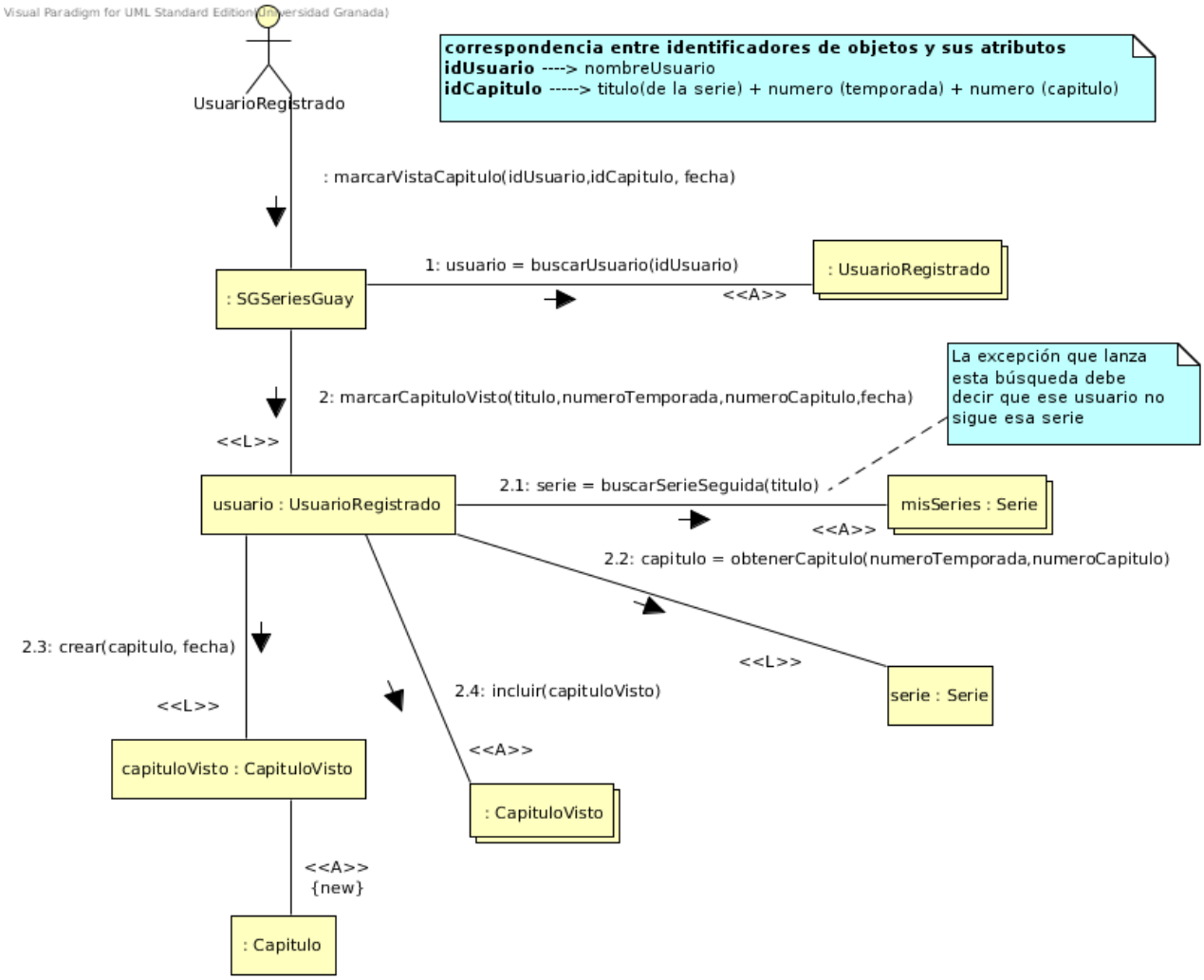
C) Valorar una serie por parte de un un usuario registrado en el sistema.



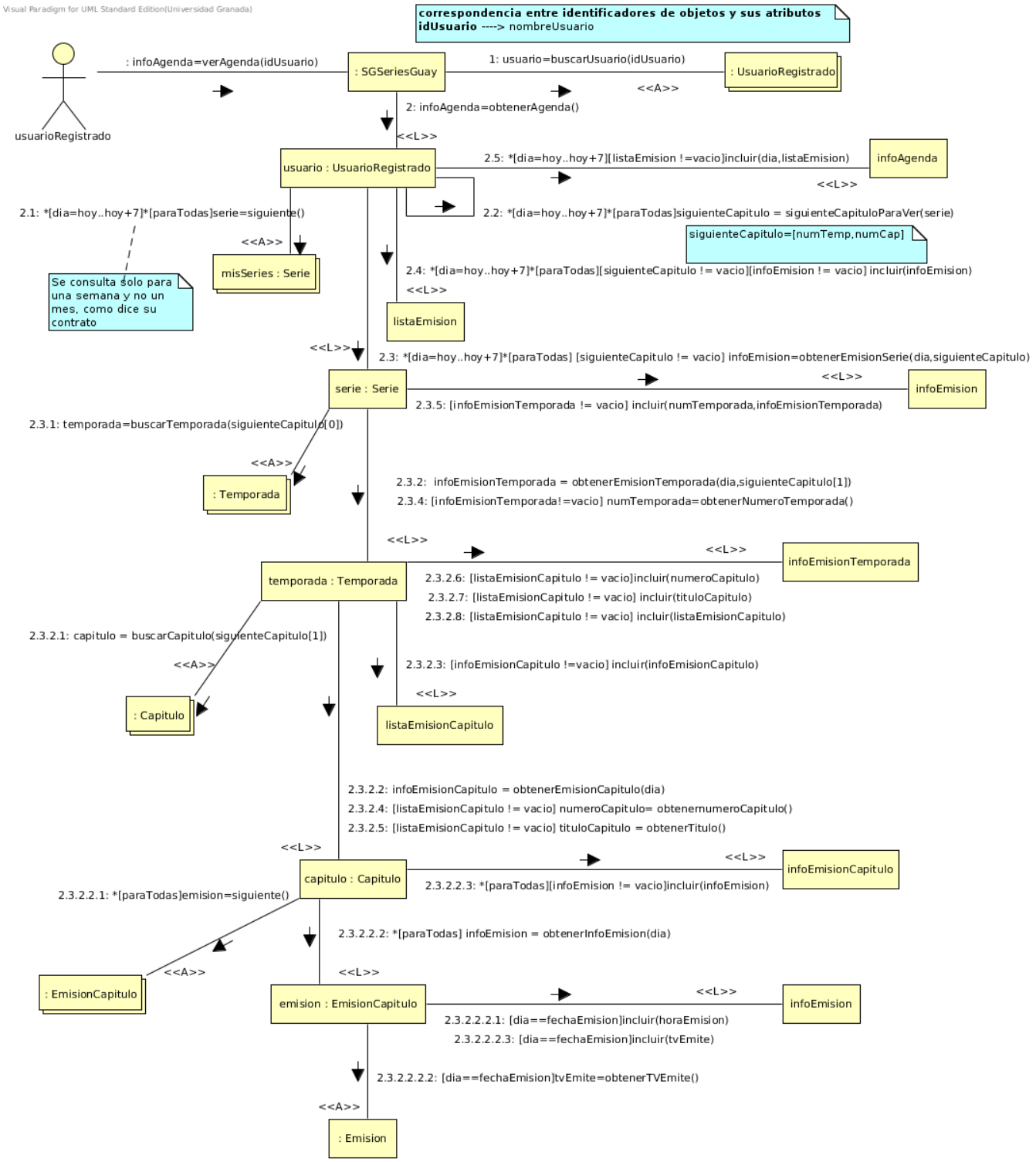
D) Definir la emisión de un capítulo de una serie



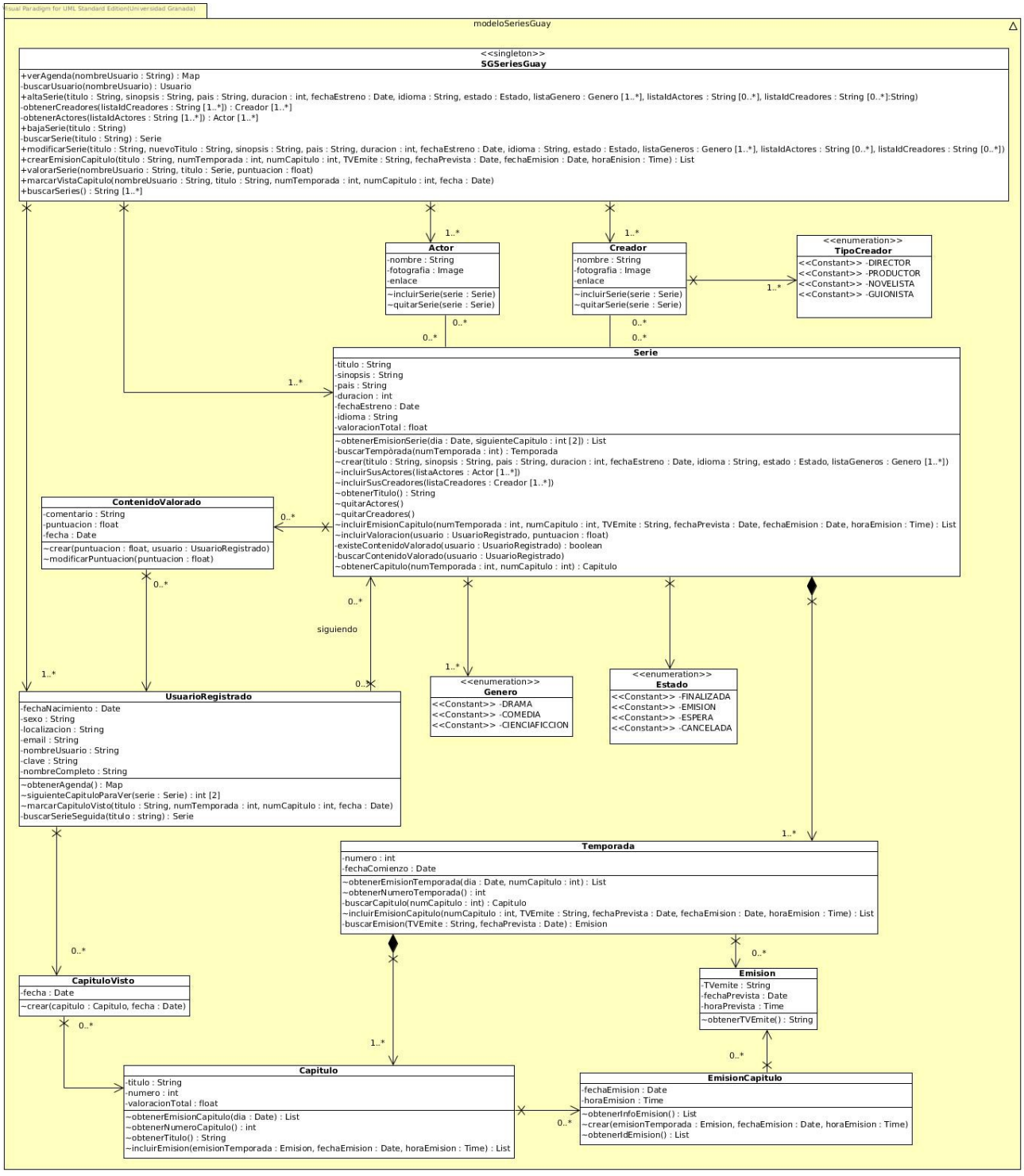
E) Marcar un capítulo como visto por un usuario registrado



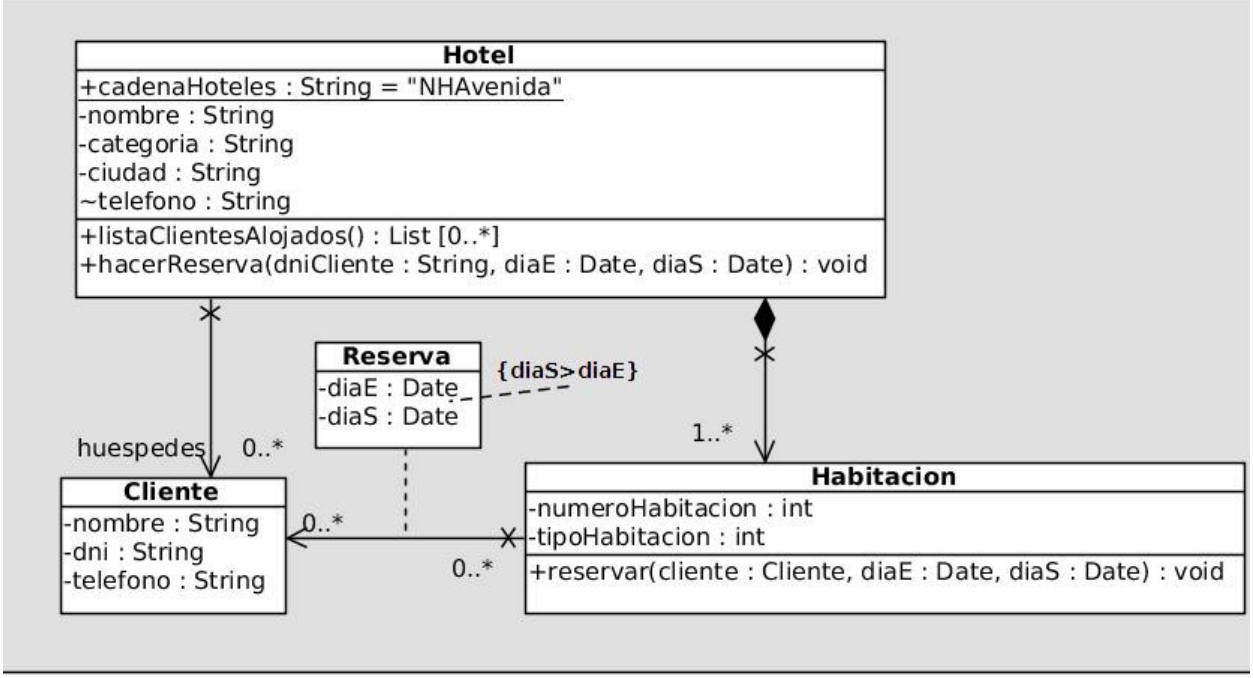
F) Consultar la agenda semanal de un usuario registrado, para ver qué capítulos se emiten de las series que está siguiendo.



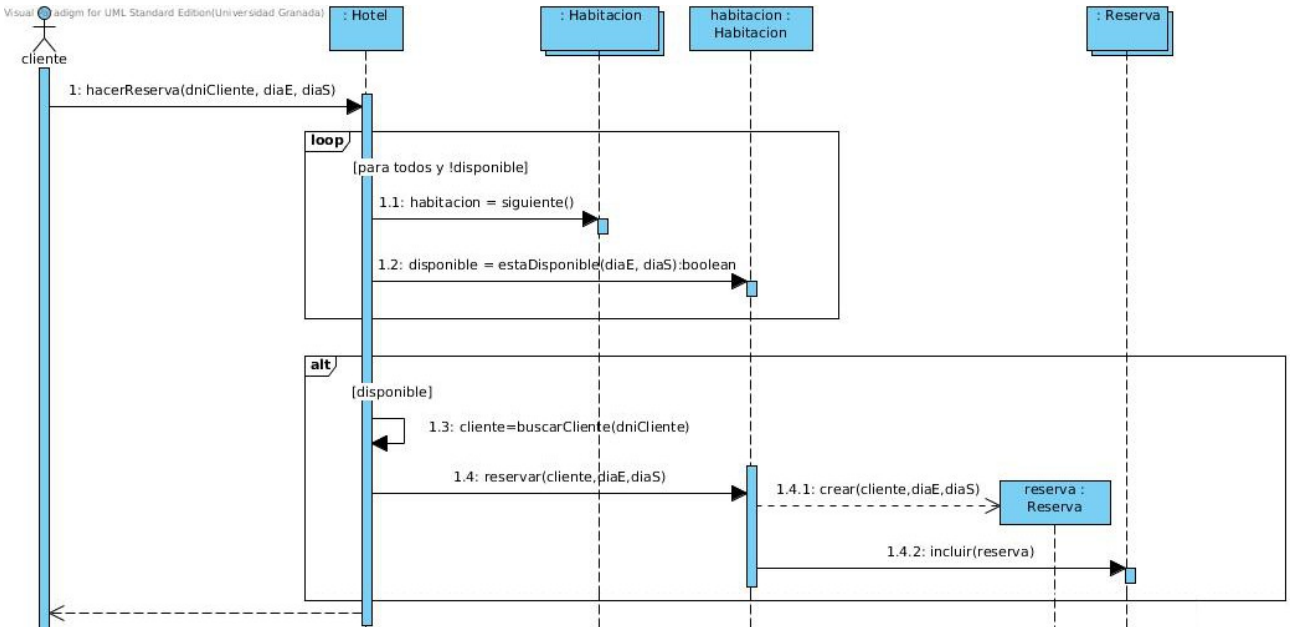
Como ayuda para la implementación de estos diagramas se proporciona el Diagrama de Clases.



Ejercicio 17 El siguiente Diagrama de clases se corresponde con un sistema de reservas de un hotel



El siguiente diagrama de secuencia se corresponde con la operación hacer una reserva en un hotel, reservando la primera habitación que se encuentre libre para las fechas indicadas.

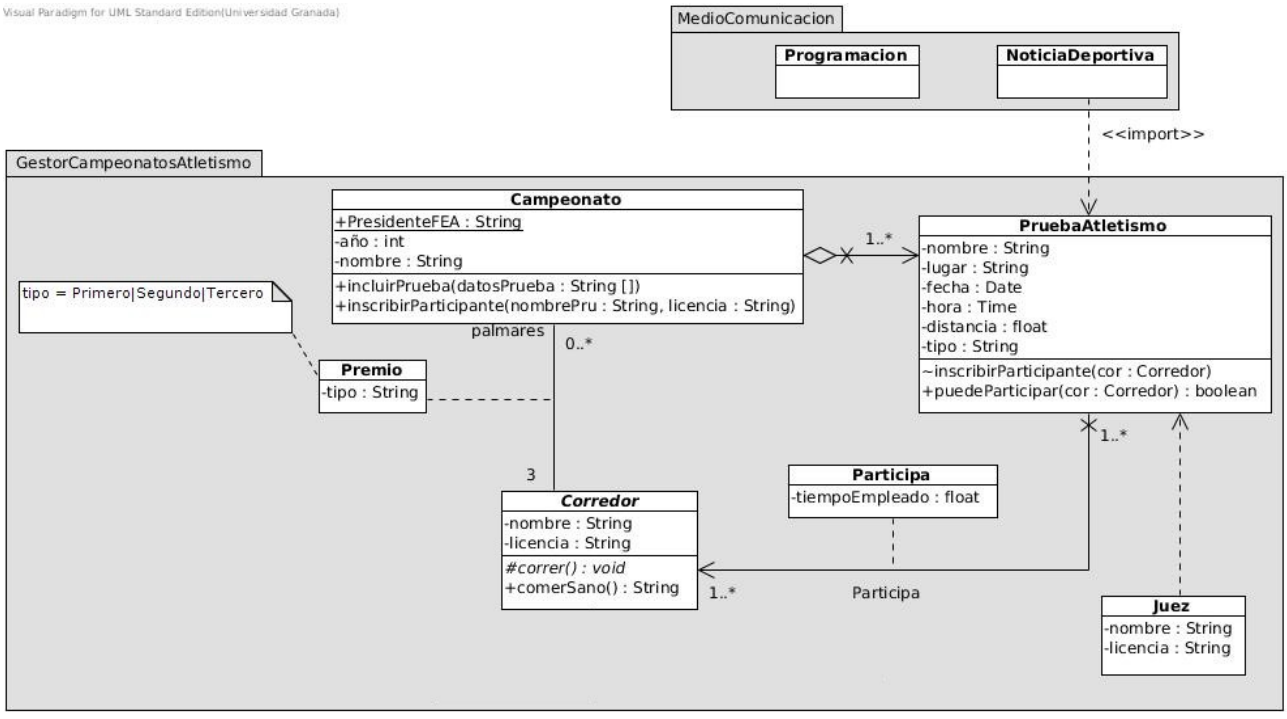


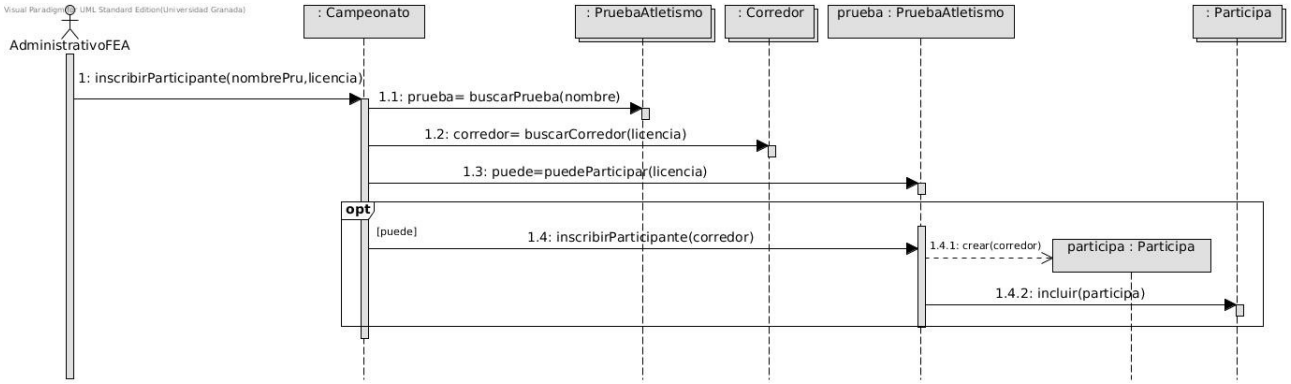
- **Responde verdadero (V) o falso (F)** a las siguientes cuestiones relacionadas con el diagrama de secuencia.

1.	Al implementar la clase *Reserva* hay que definir un método llamado ***crear*** con argumentos, para crear e inicializar los objetos de la clase.	
2.	Existe un canal directo de comunicación entre el objeto de la clase *Hotel* y la colección de objetos de la clase *Habitacion*	
3.	El mensaje etiquetado con 1.3 es un mensaje recursivo que debe enviarse mientras *disponible*=true	
4.	Los mensajes etiquetados con 1.1 y 1.2 se envían ambos a objetos de la clase **Habitación**	
5.	La flecha discontinua desde el objeto Hotel al actor Cliente representa un envío de mensaje asíncrono	

- Completa el diagrama de clases incluyendo los métodos que falten a la vista del diagrama de secuencia.
- Implementa en **Java y Ruby** el método **hacerReserva(...)** de la clase **Hotel**, siguiendo el diagrama de secuencia proporcionado y ayudándote del diagrama de clases.
- Implementa en **Java y Ruby** el método **reservar(...)** de la clase **Habitación**.
- Implementa en **Ruby** la clase **Habitación** con todos sus atributos y métodos, siguiendo los diagramas proporcionados.

Ejercicio 18 Dados los siguientes diagramas:

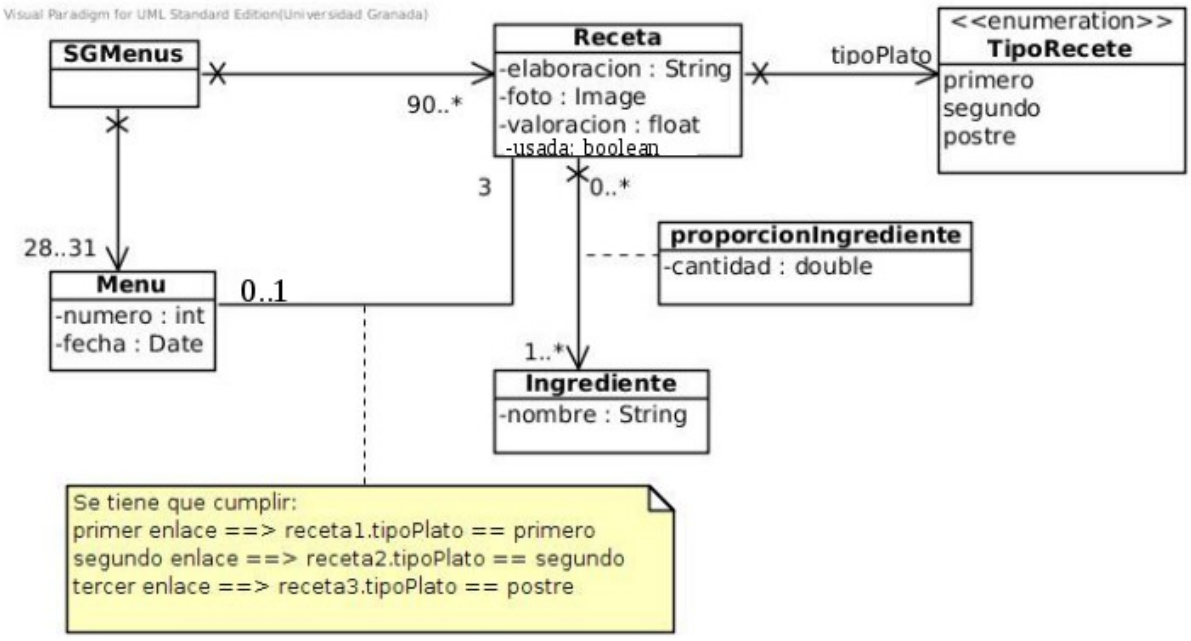




- Implementa el diagrama de clases en Java y Ruby
- Implementa el diagrama de secuencia completo en Java y Ruby

EJERCICIOS COMPLEMENTARIOS

Ejercicio 19 (Problema resuelto). Partiendo del siguiente diagrama de clases, que se corresponde con una de las posibles soluciones al ejercicio 12.A de la relación de problemas del tema 2.2:

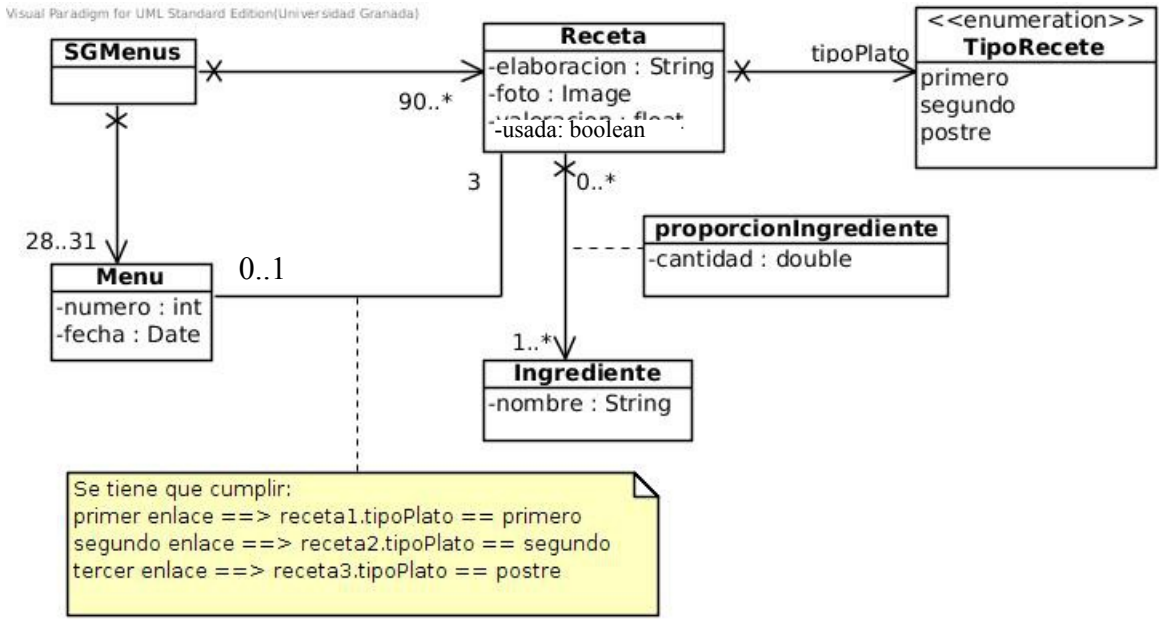


Obtener el diagrama de interacción (secuencia o comunicación) de la siguiente operación:

Incluir un nuevo menú en el sistema, de tal forma que el número de menú y la fecha serán los siguientes al del último menú definido. Un menú está compuesto por un primer plato, un segundo y un postre. Éstos son elegidos de forma secuencial entre las recetas que existen en el sistema, teniendo en cuenta que los menús que se están definiendo para el mes no pueden repetir recetas entre ellos. Una vez finalizada la operación se ha creado un objeto menú y se ha enlazado con tres objetos receta: primer plato, segundo plato y postre. Se asume que en la clase Menu existe un método obtenerDatos() que devuelve el número de menú y la fecha.

Solución del ejercicio 19:

Partiendo del siguiente esquema de diagrama de clases y de la descripción proporcionada de la operación:

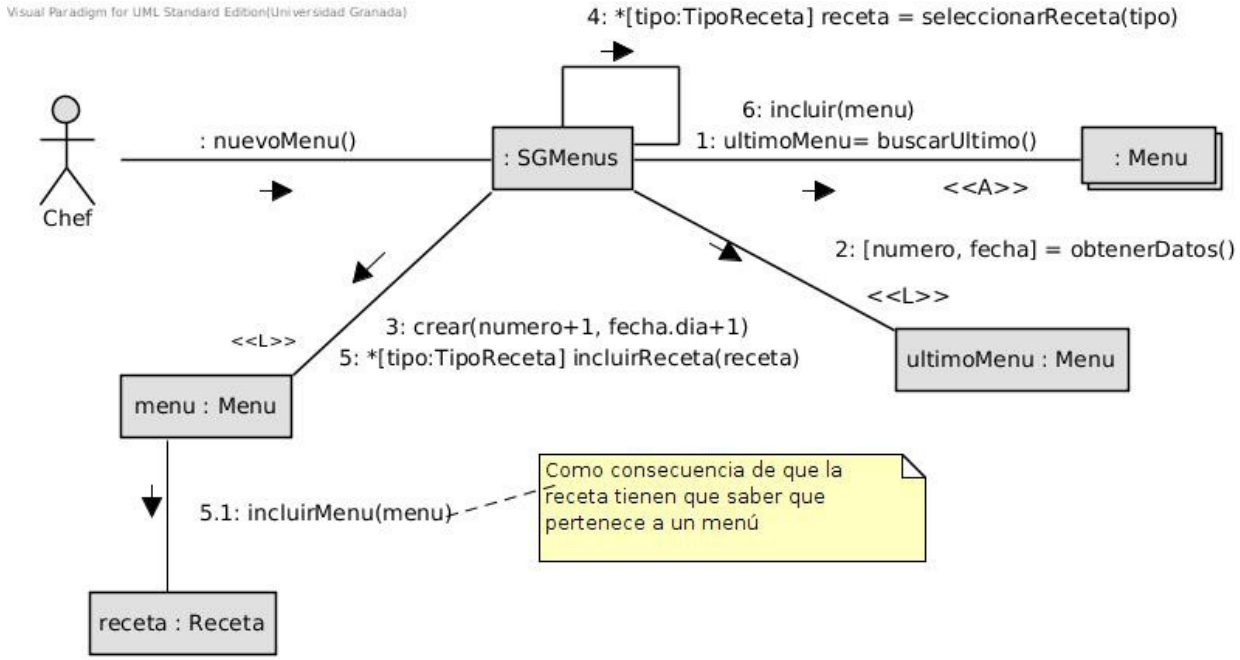


Seguimos el siguiente procedimiento:

1. Identificar parámetros de la operación:
   operación que no necesita ningún parámetro todo lo que necesita ya está incluido en el sistema, la operación quedaría: nuevoMenu()
2. Identificar objeto responsable: SGMenus
   Representar primer nivel de envío de mensaje.

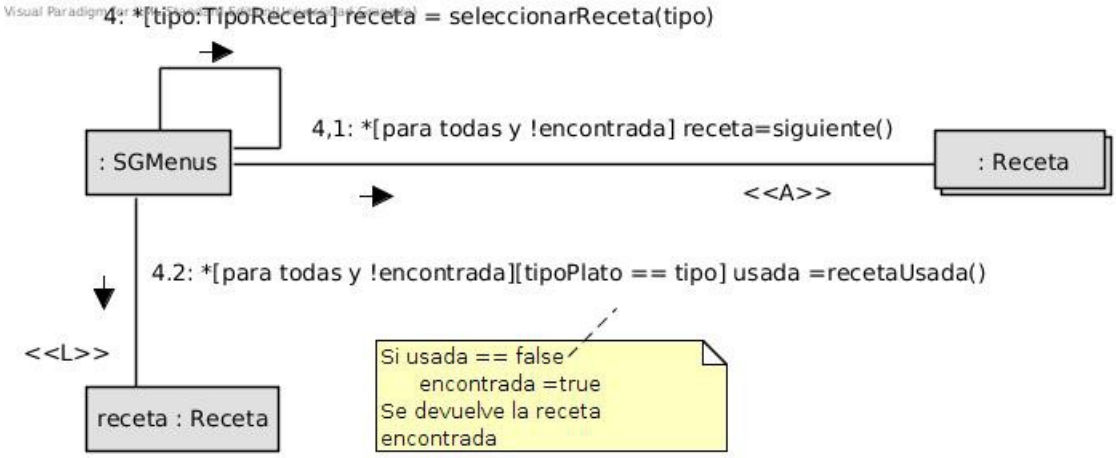
3. Identificar responsabilidades de :SGMenus
   a) Buscar el último menú, para obtener su número y fecha.
   b) Crear el nuevo menú a partir de estos datos.
   c) Seleccionar una receta para el primer plato, teniendo en cuenta lo indicado anteriormente y enlazarla con el menú.
   d) Seleccionar una receta para el segundo plato, teniendo en cuenta lo indicado anteriormente y enlazarla con el menú.
   e) Seleccionar una receta para el postre plato, teniendo en cuenta lo indicado anteriormente y enlazarla con el menú.
   f) Incluir el nuevo menú dentro de la lista de menús mensuales.

4. Primer nivel de subordinación:



5. Segundo nivel de subordinación, que se corresponde con el envío de mensaje seleccionarReceta() y responsabilidad de :SGMenu

   - Responsabilidad de :SGMenu
     1. Recorrer la lista de recetas.
     2. Determinar si la receta en cuestión es del tipo que estamos buscando y no ha sido usada en otro menú.
     3. En el caso que se cumpla esto último, devolverla como receta valida



Ejercicio 20. A partir de la solución del ejercicio 19:

- Refina el modelo obtenido, encontrando otras posibles soluciones.
- Pasa a diagrama de secuencia el diagrama de comunicación obtenido.
- Implementa el diagrama en Java y Ruby.

Ejercicio 21. Partiendo de los diagramas de clases obtenidos en la relación de problemas del tema 2.2 en el ejercicio 12 de los supuestos B a G, obtener el diagrama de interacción (secuencia o comunicación) de las siguientes operaciones:

A (del 12.B) Obtener los resultados de la carrera celebrada en una determinada fecha y lugar, se debe proporcionar el resultado por equipos e individual por atletas medallas de oro, plata y bronce. Como salida se debe proporcionar fecha, lugar y categoría de la carrera, para los equipos ganadores, el nombre del equipo, el tiempo invertido y el nombre de los atletas que lo componen y para los resultados individuales, el nombre del atleta y el tiempo invertido.

B (del 12.C) Matricular a un alumno de una asignatura en un grupo concreto, la asignatura es identificada por un código y el grupo por un letra que es su denominación. Terminada la operación se ha enlazado un objeto alumno con el grupo de una asignatura.

C (del 12.D) Hacer una reserva de una habitación de un hotel por un cliente para unos determinados días. Las disponibilidad de la habitación ya fue comprobada previamente. Terminada la operación se ha creado un objeto reserva y enlazado con habitación y con cliente. Previamente se ha dado de alta al cliente si no existía en el sistema.

D (del 12.E) Programar un evento asignándole una sala en la que se va a desarrollar el evento. Para ello hay que proporcionar el nombre del evento y las fechas en las que se va a celebrar, a partir de esas fechas hay que buscar una sala libre, una vez terminada la operación se ha creado un objeto evento y se ha enlazado con la sala en la que se va a llevar a cabo, si no se encuentran salas disponibles se debe producir una excepción, indicando lo que pasa.

E (del 12.F) Mover un disco de una varilla origen a otra destino, si el disco que hay en la parte superior de la varilla destino es menor que en disco que se va a mover hay que proporcionar una excepción y no permitir el movimiento del disco.

F (del 12.G) Incluir un nuevo polígono proporcionando sus vértices y comprobando que es un polígono regular, si no lo es, se debe proporcionar una excepción y no permitir su construcción. Una vez terminada la operación de ha creado un objeto polígono regular y tantos objetos punto como vértices se hayan proporcionado y enlazado el polígono con sus vértices.