Tema 2: Lección 3

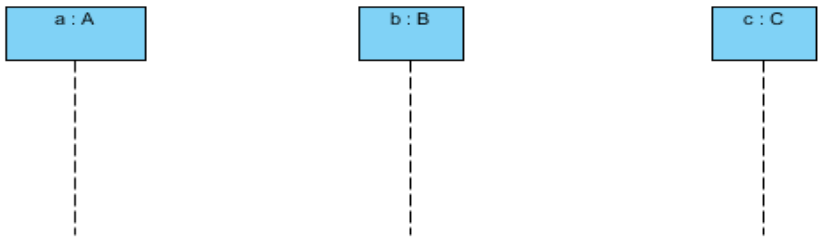
Ejercicio 1. En general, ¿qué representan los diagramas de interacción de UML? ¿Cuáles son sus componentes principales?

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.

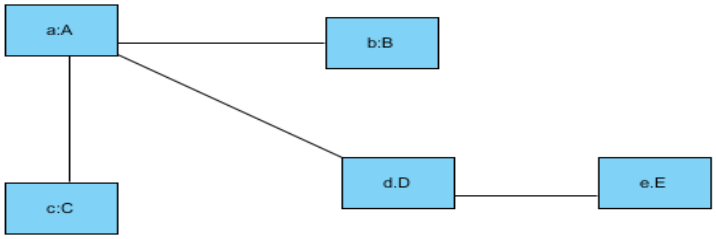
Ejercicio 3. ¿Qué relación existe entre el diagrama de clases y los diagramas de interacción?

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

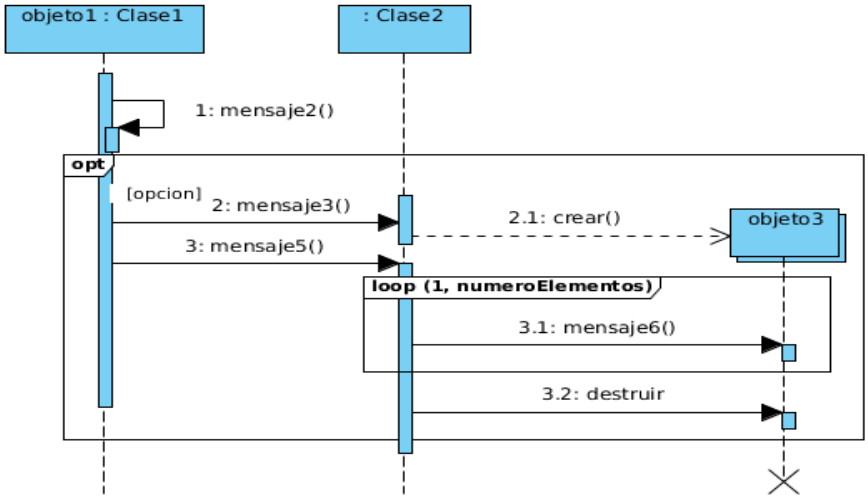
Tradúcelo a su correspondiente diagrama de comunicación e impleméntalo en Java y Ruby.



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.



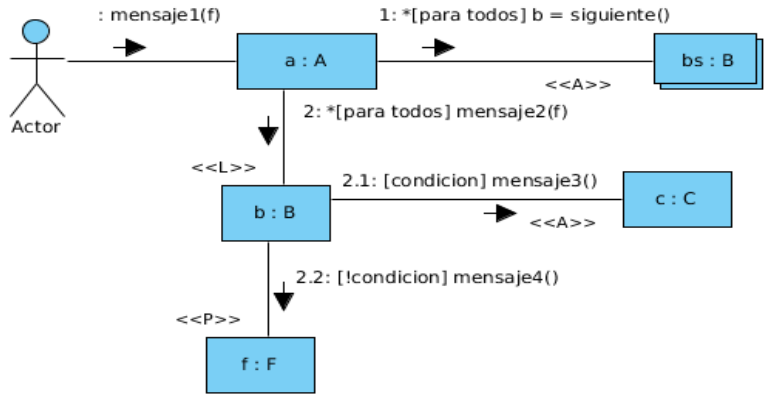
Ejercicio 6: Dado el siguiente Diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones



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	

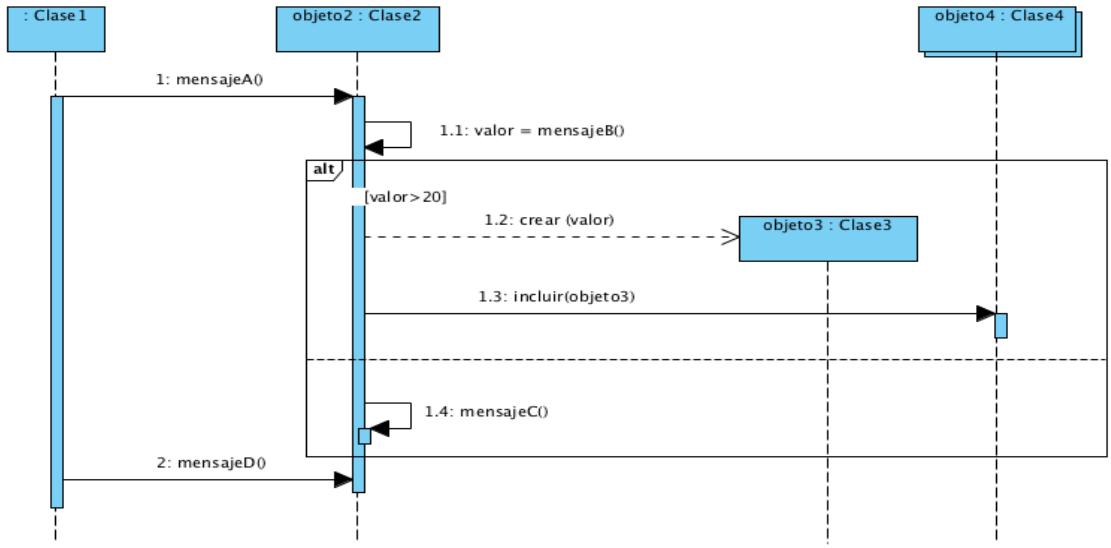
Ejercicio 7: Dado el siguiente Diagrama de comunicación, responde verdadero (V) o falso (F) a las siguientes cuestiones



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]

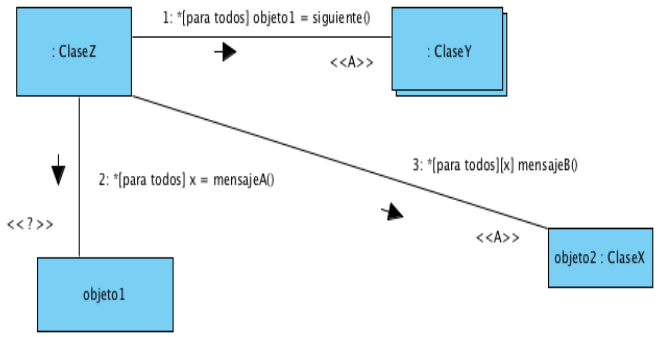
El siguiente código Java **es correcto**:
public class A {
public void mensaje1(F f){
for(B b:bs){
b.mensaje2(f);
}
}
}

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



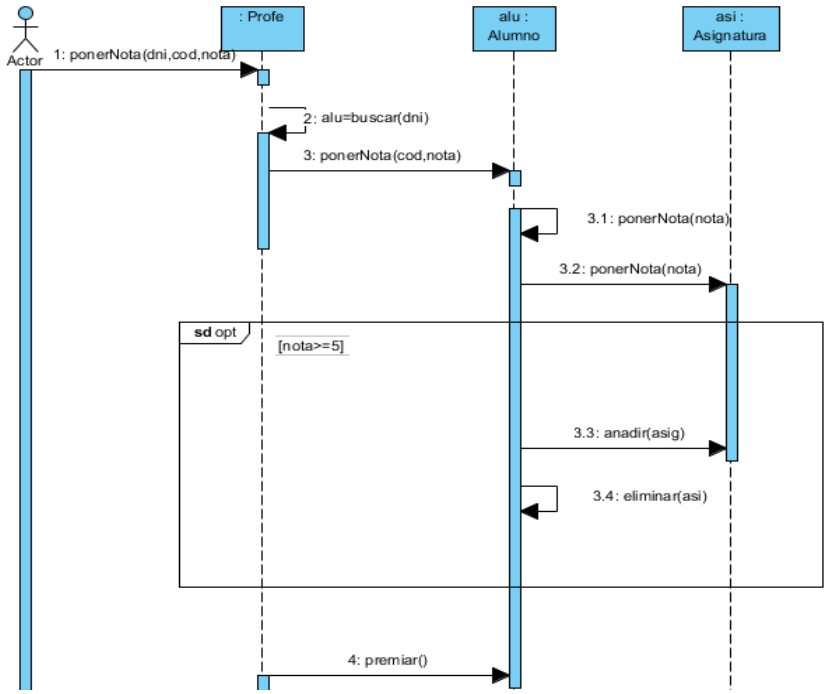
La clase **Clase2** debe tener un método que se llame **mensajeB**.
El paso numerado con un 2 debería ser 1.5.
En el diagrama de comunicación equivalente, el tipo de enlace entre el **objeto2** y el **objeto3** sería **<<A>>**.
El paso 1.2 corresponde a una llamada al constructor por defecto de la clase **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



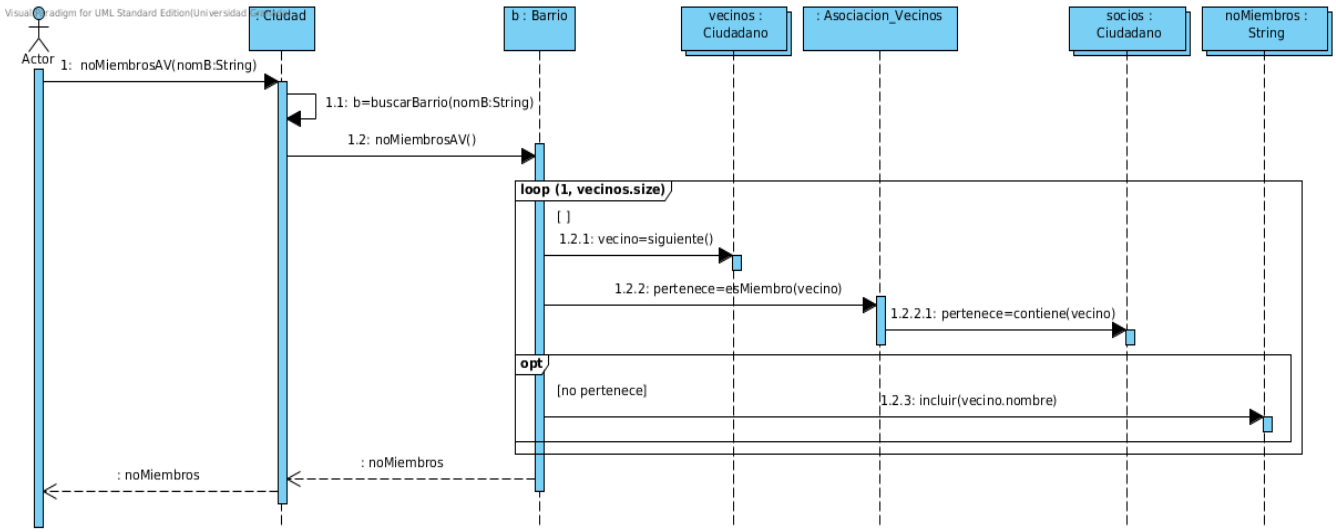
El tipo de enlace que aparece con interrogación no puede ser **<<A>>**.	
Al codificar el diagrama, las instrucciones correspondientes a los pasos 1, 2 y 3 estarán todas dentro de un mismo bucle for.	
Aunque no se indique explícitamente, es posible conocer la clase de **objeto1** por la información contenida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

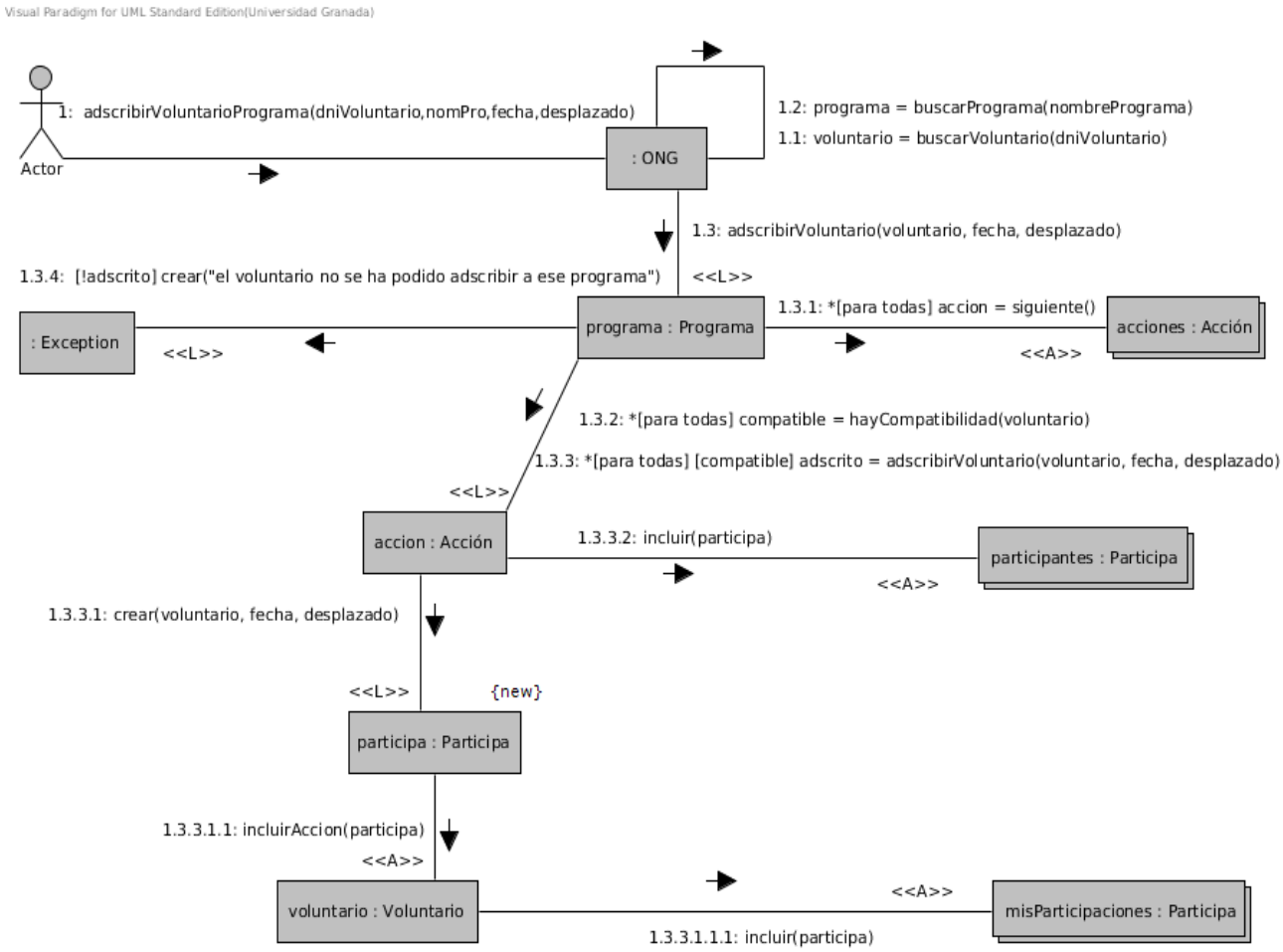
Ejercicio 11. A partir del siguiente diagrama de secuencia:



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:



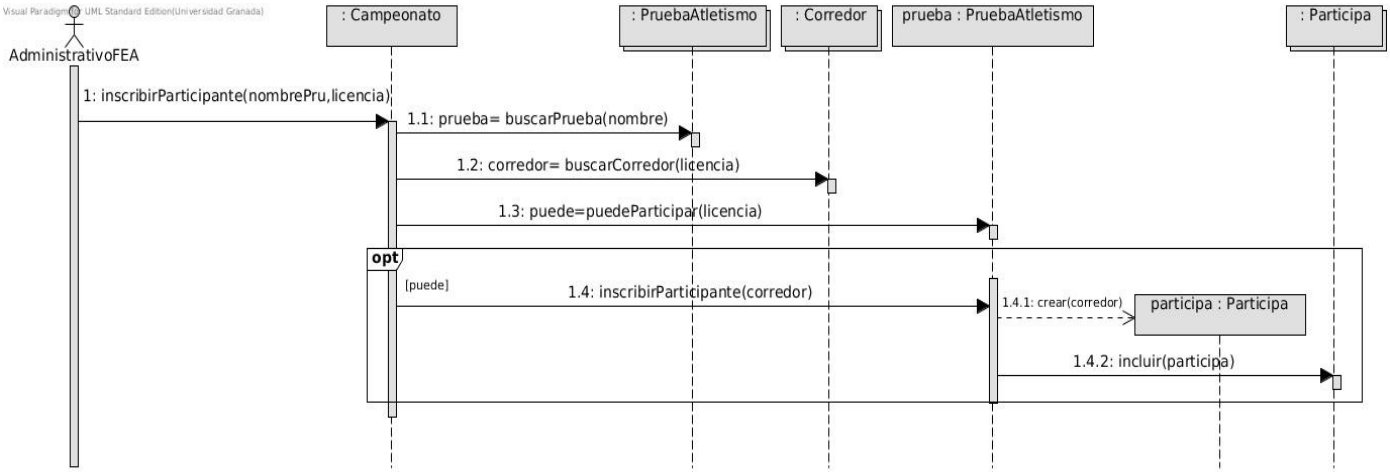
A) Responde V(verdadero) o F(falso) a las siguientes cuestiones:

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



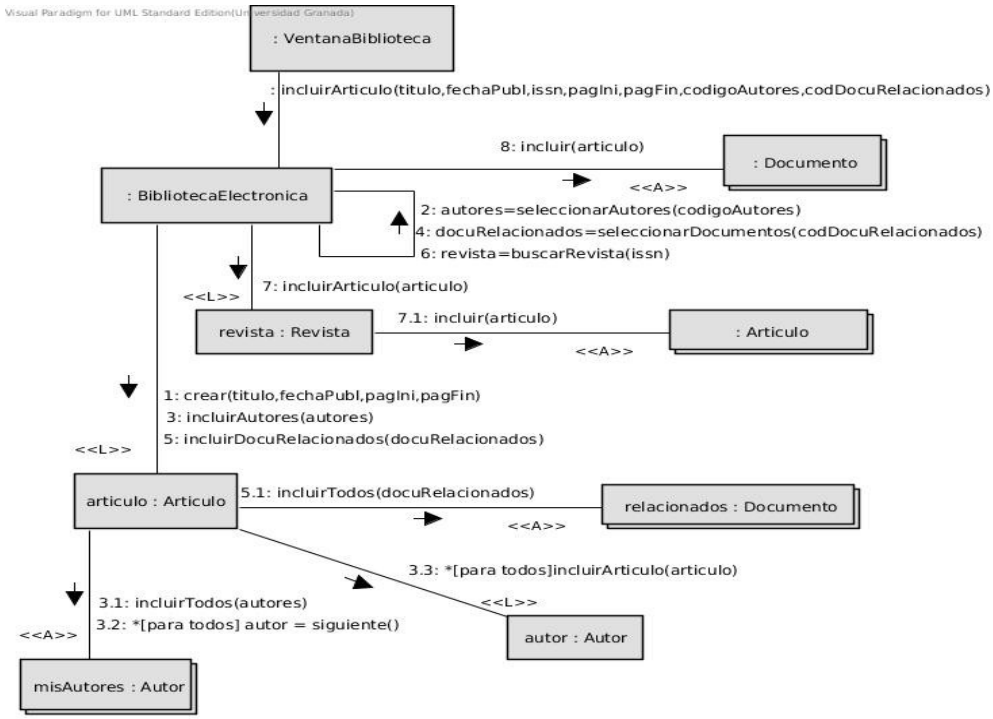
A) Responde V (verdadero) o F (Falso) a las siguientes cuestiones:

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**	

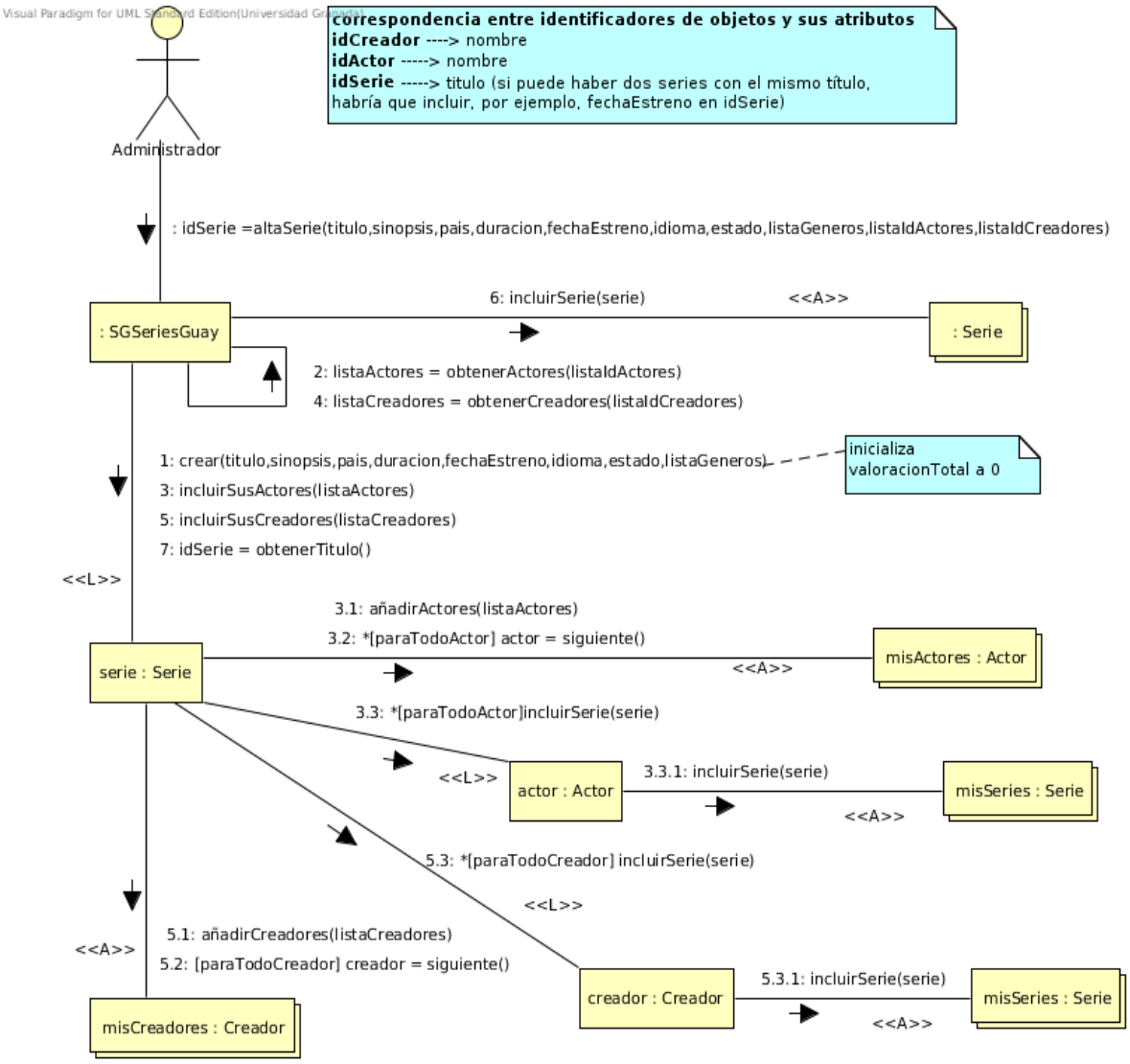
B) Impleméntalo en java y el Ruby

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

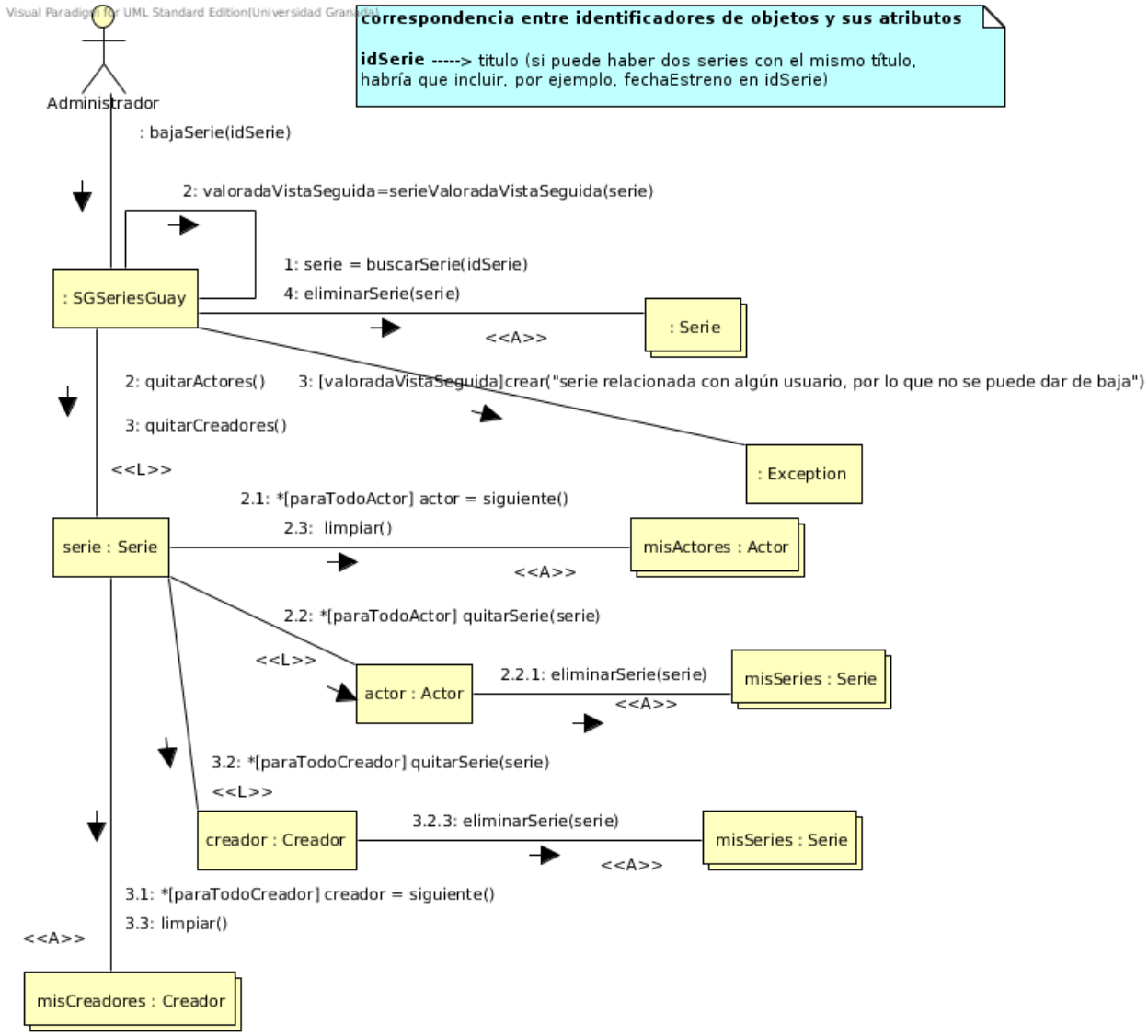
Ejercicio 15 .Traduce los diagramas de la práctica 3 de secuencia a comunicación o de comunicación a secuencia según corresponda.

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.

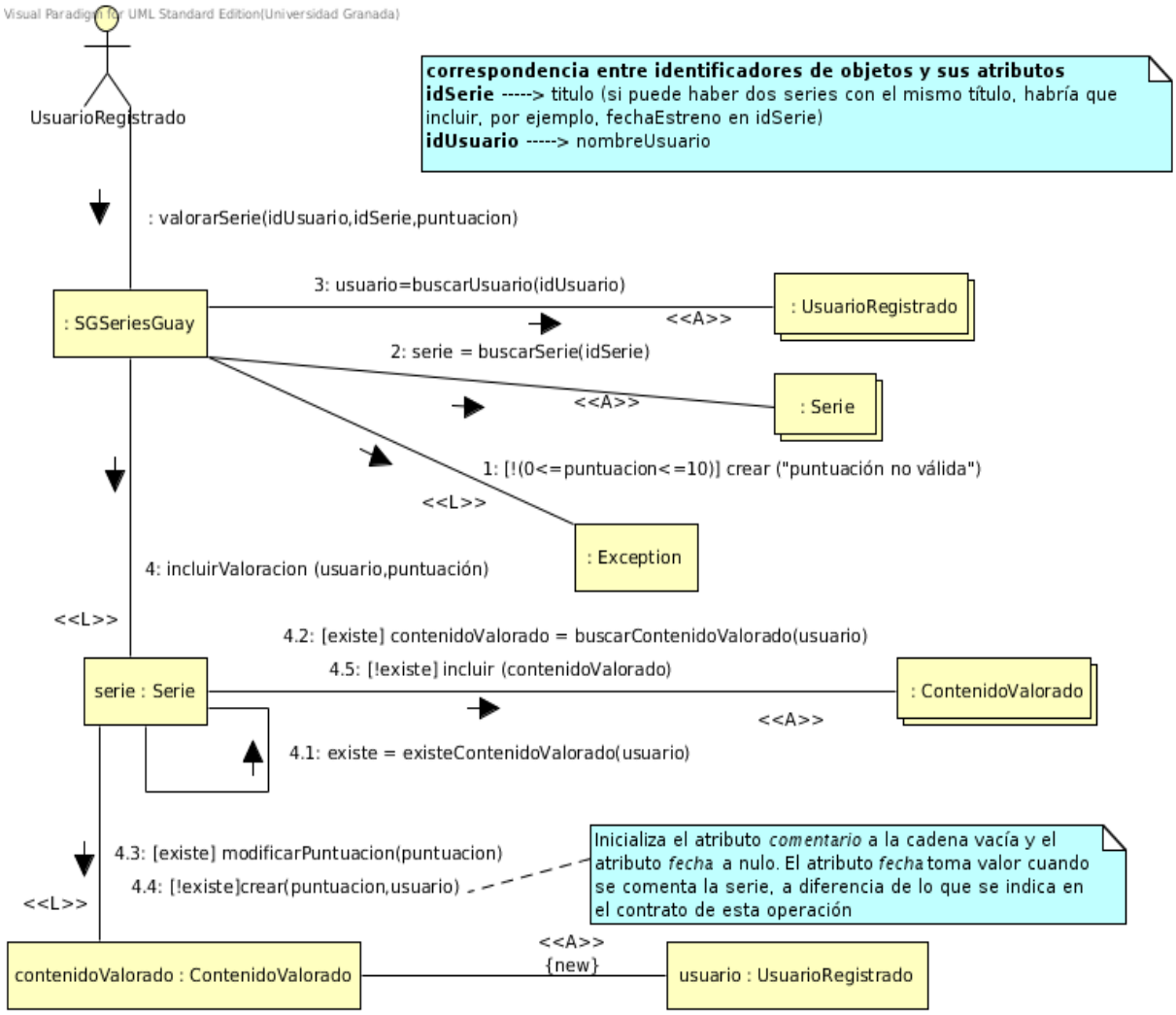
A) Incluir una nueva serie de TV en el sistema **:SGSeriesGuay**


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

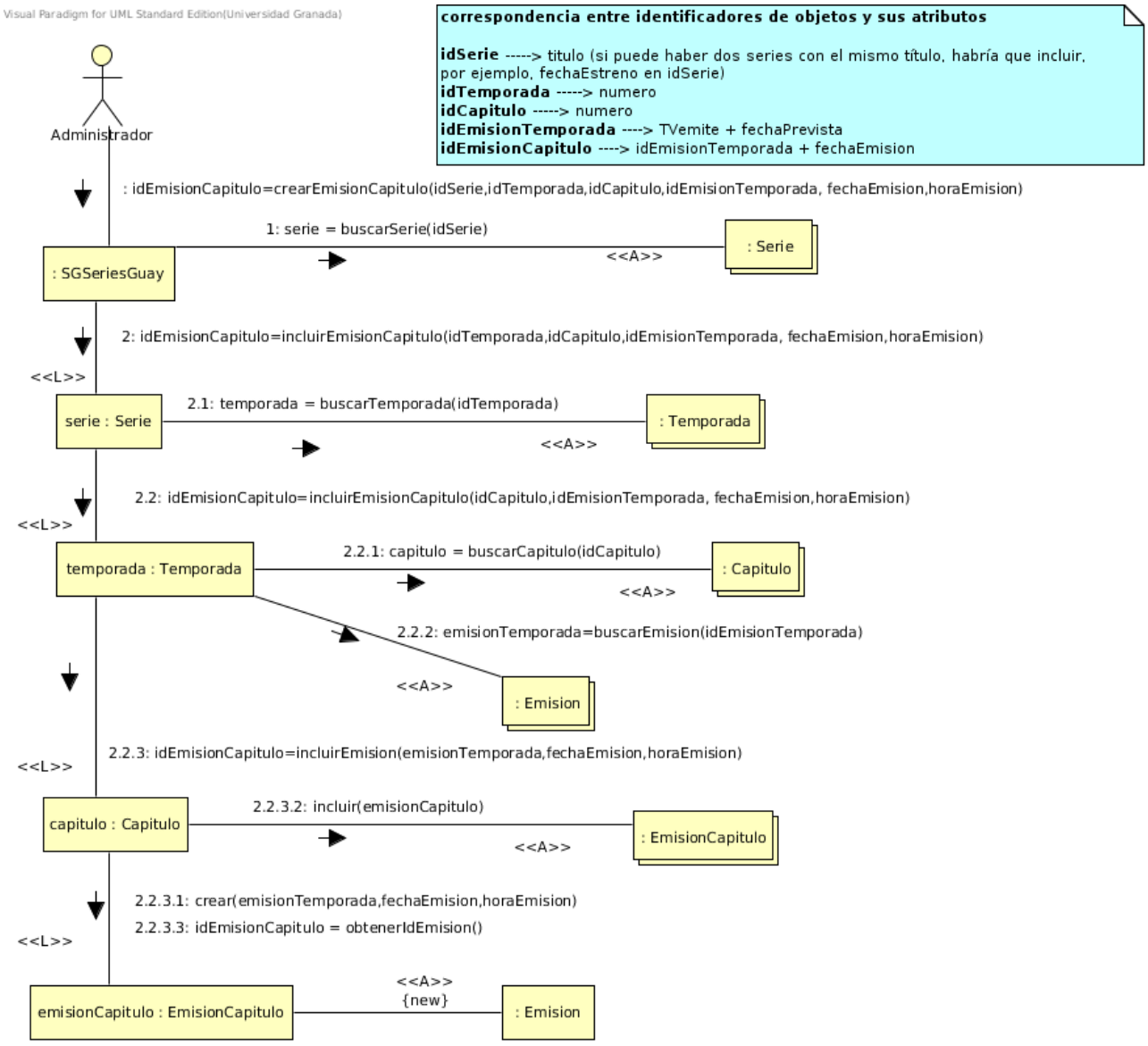
B) Eliminar una serie



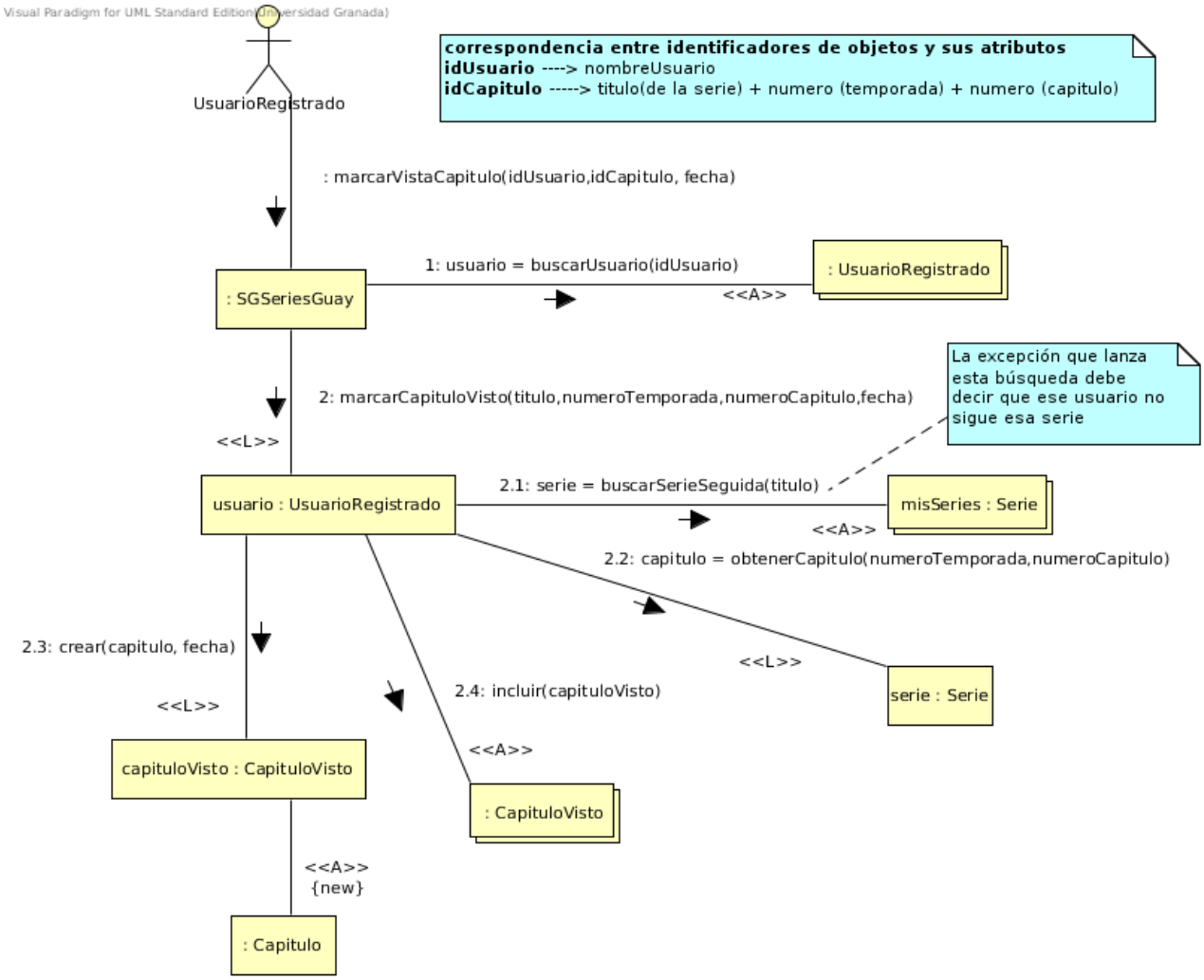
C) Valorar una serie por parte de un un usuario registrado en el sistema.



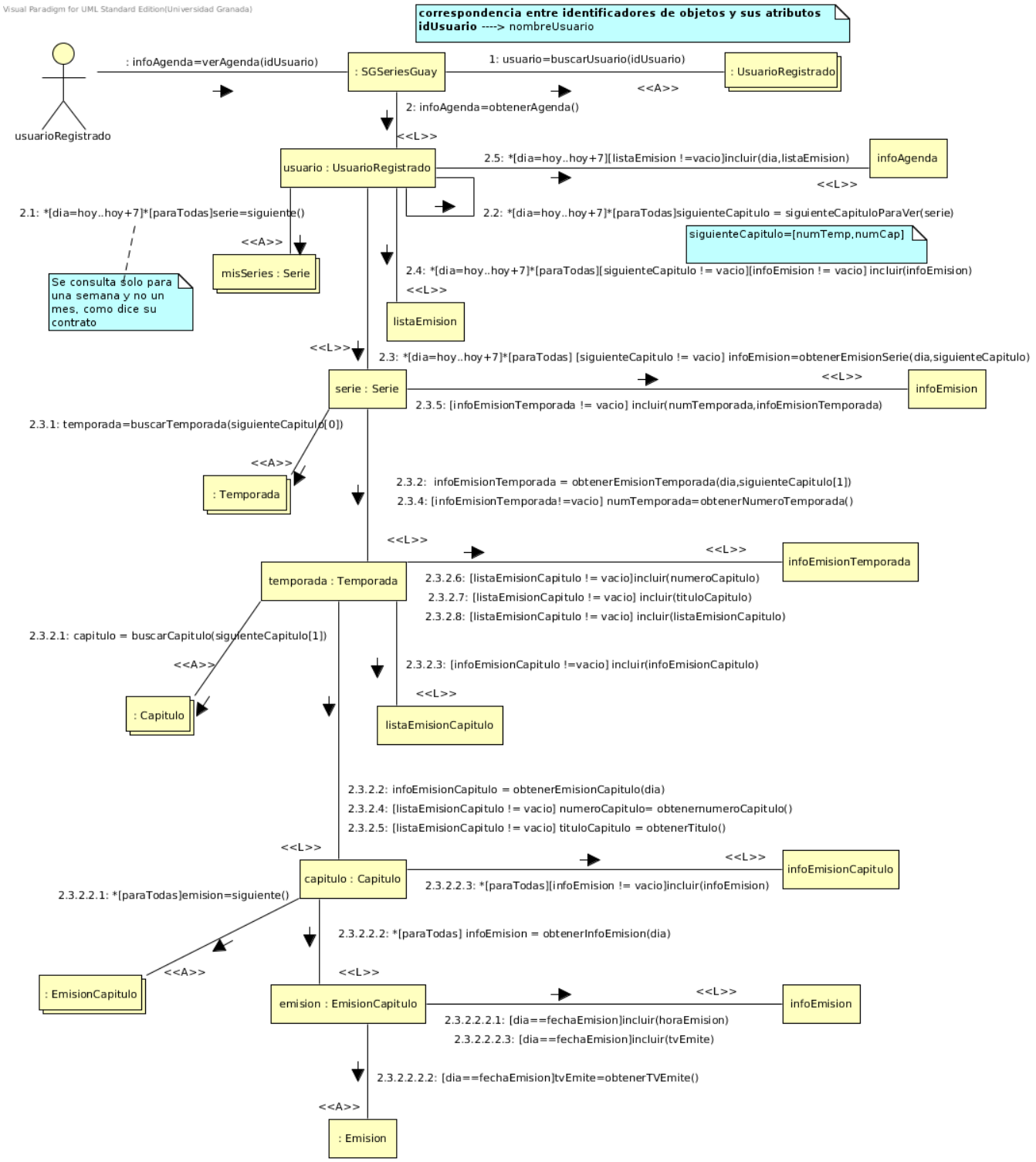
D) Definir la emisión de un capítulo de una serie



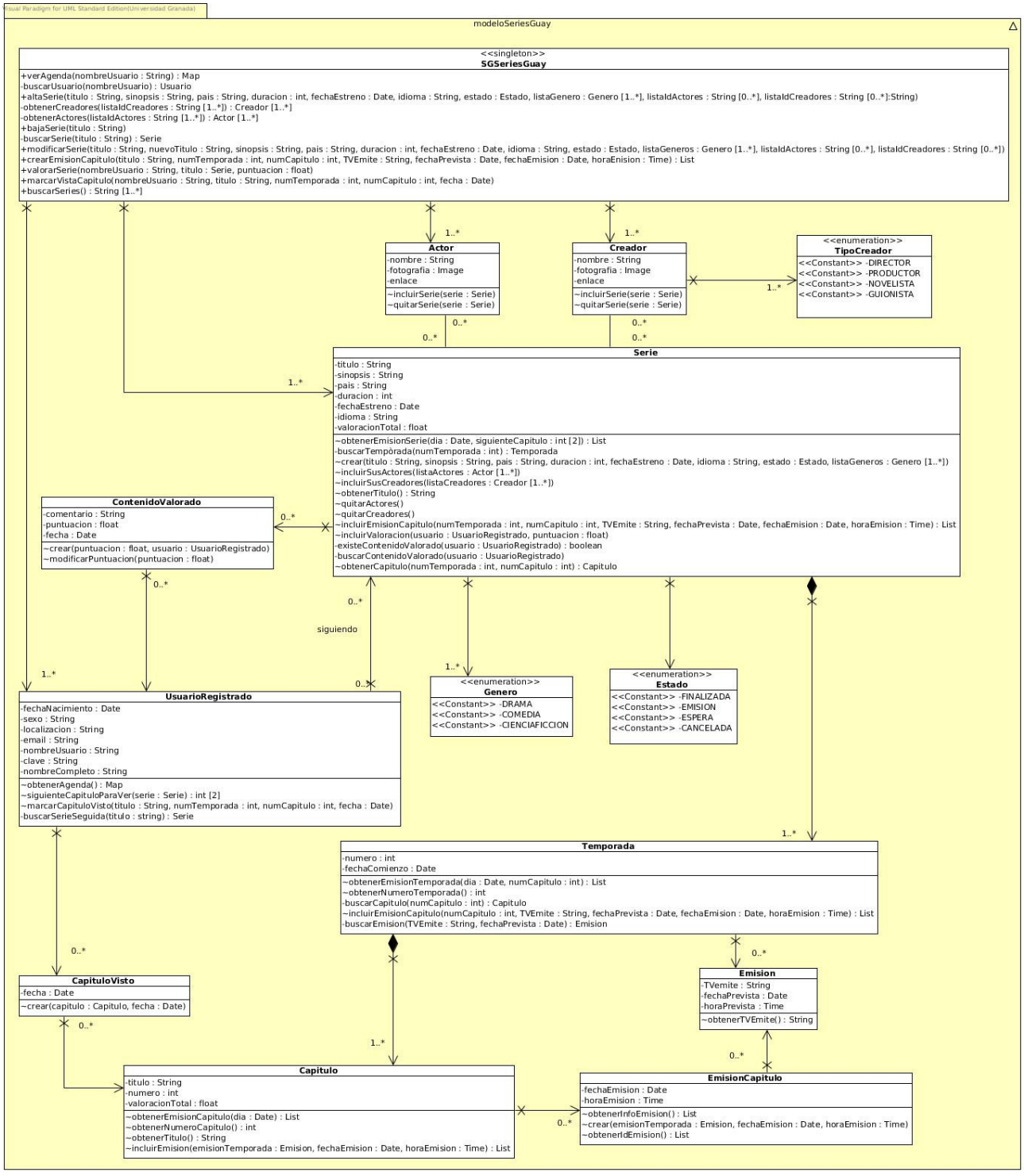
E) Marcar un capítulo como visto por un usuario registrado



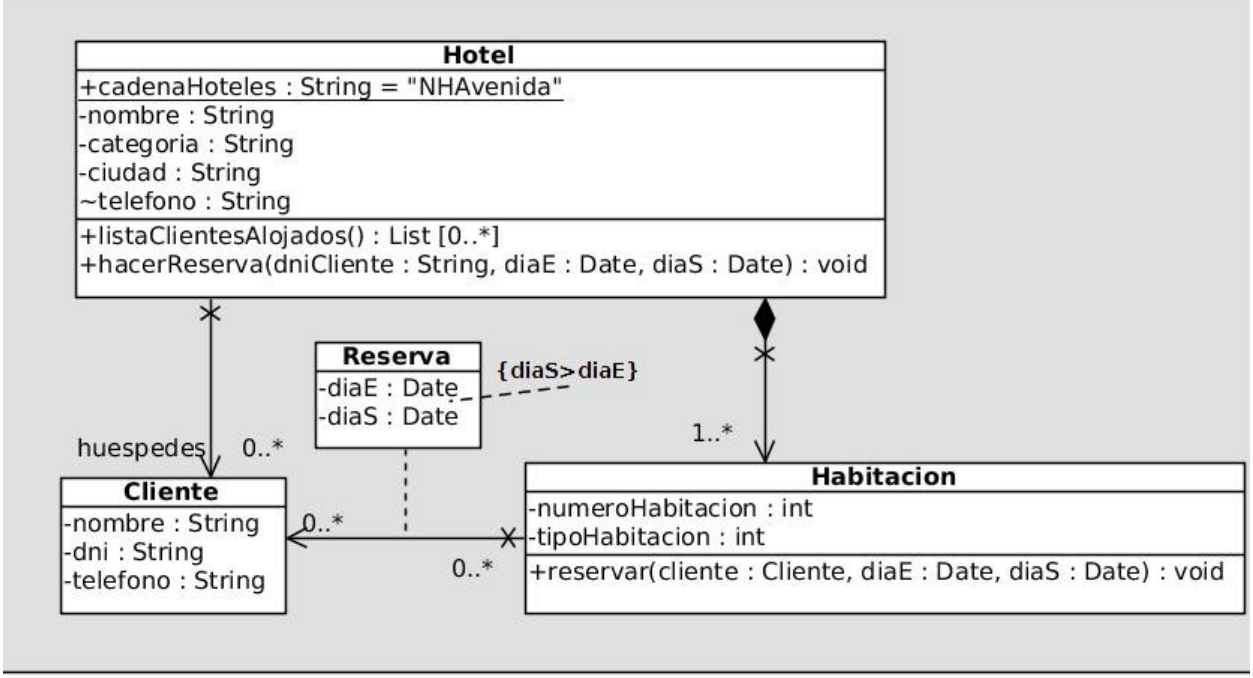
F) Consultar la agenda semanal de un usuario registrado, para ver qué capítulos se emiten de las series que está siguiendo.



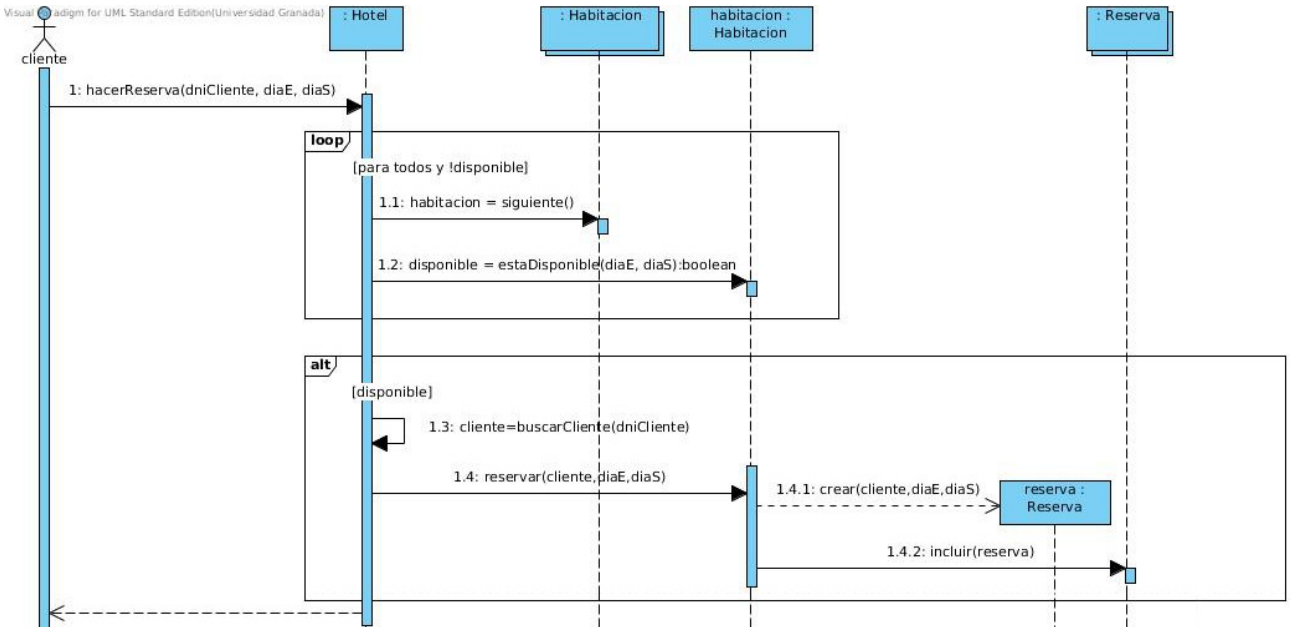
Como ayuda para la implementación de estos diagramas se proporciona el Diagrama de Clases.



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.

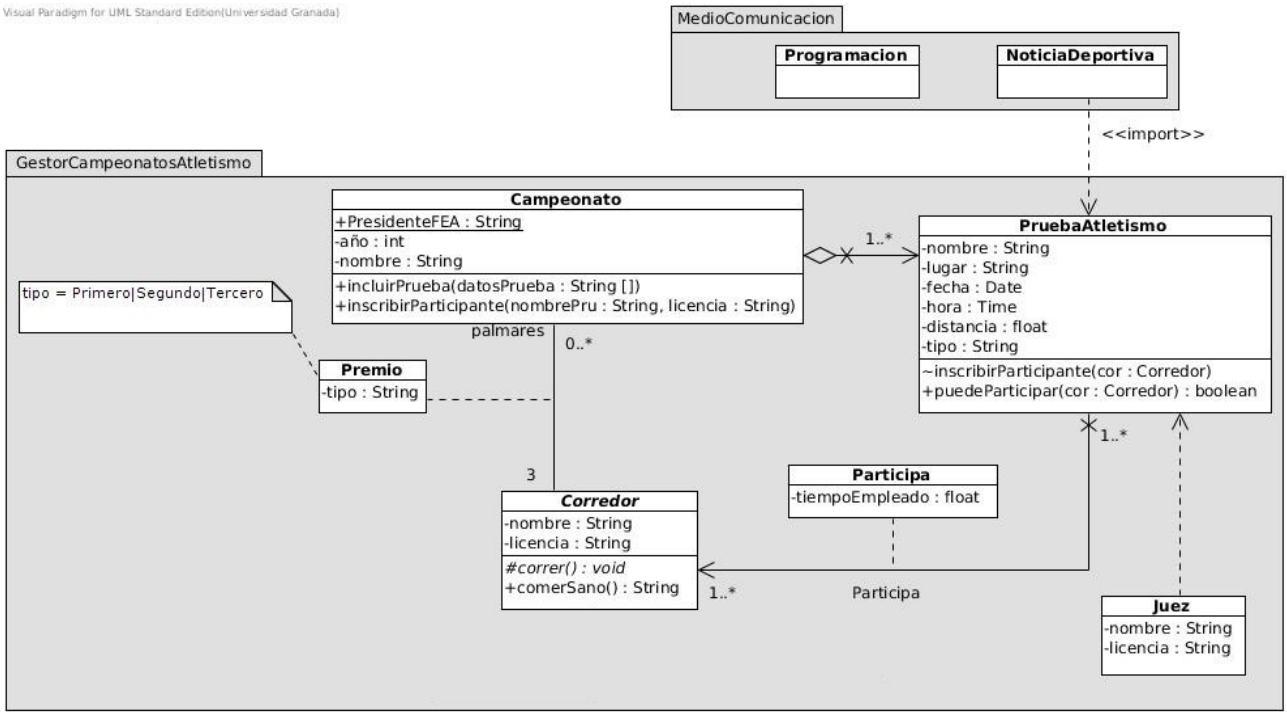


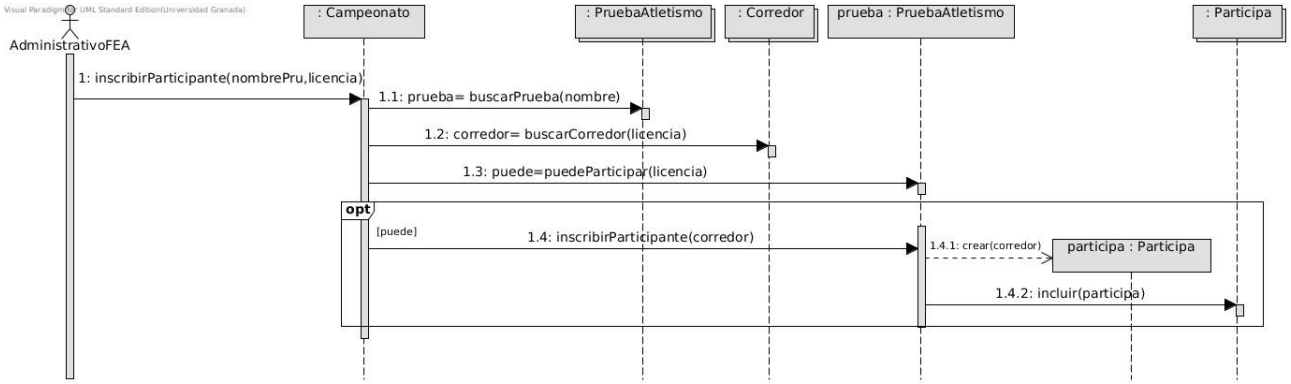
- **Responde verdadero (V) o falso (F)** a las siguientes cuestiones relacionadas con el diagrama de secuencia.

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:

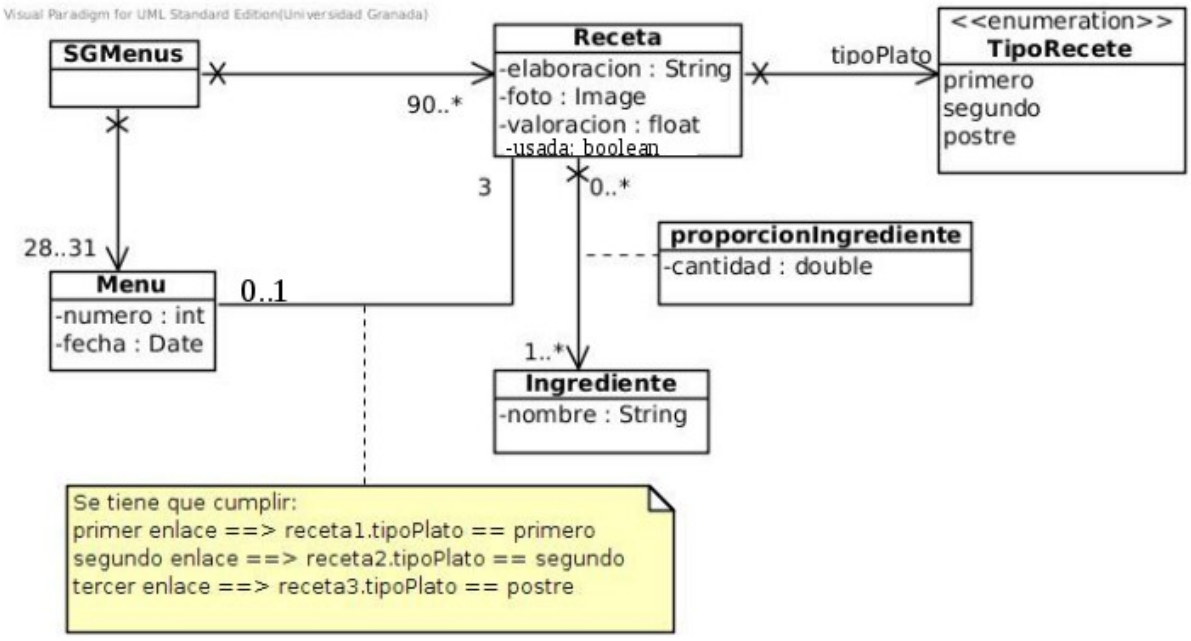




- Implementa el diagrama de clases en Java y Ruby
- Implementa el diagrama de secuencia completo en Java y Ruby

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:

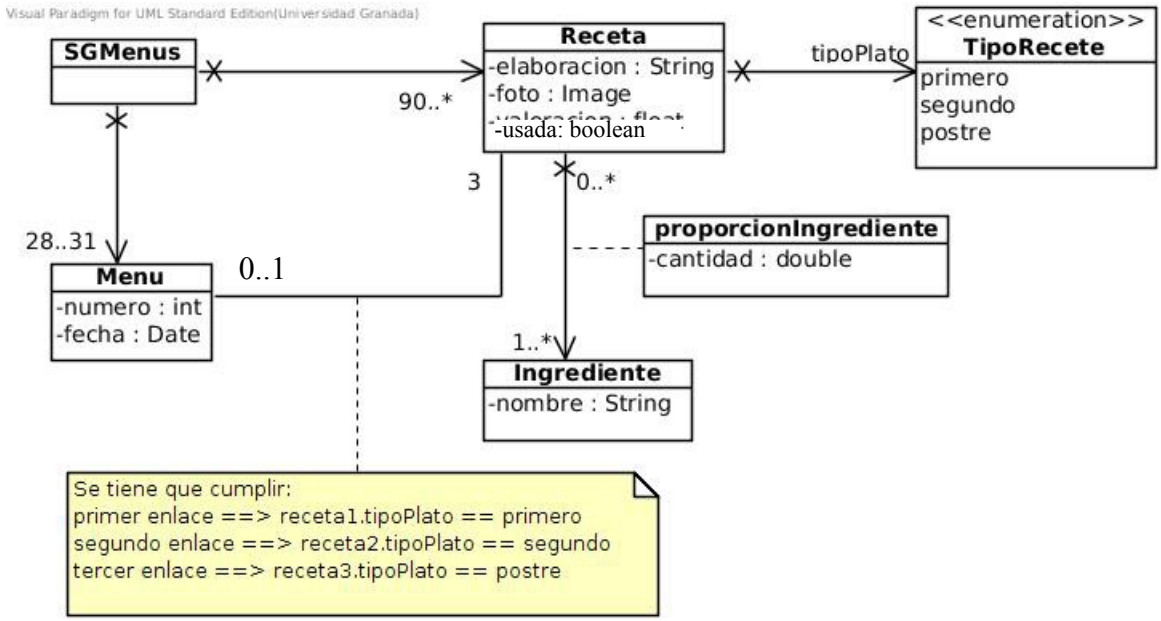


Obtener el diagrama de interacción (secuencia o comunicación) de la siguiente operación:

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:

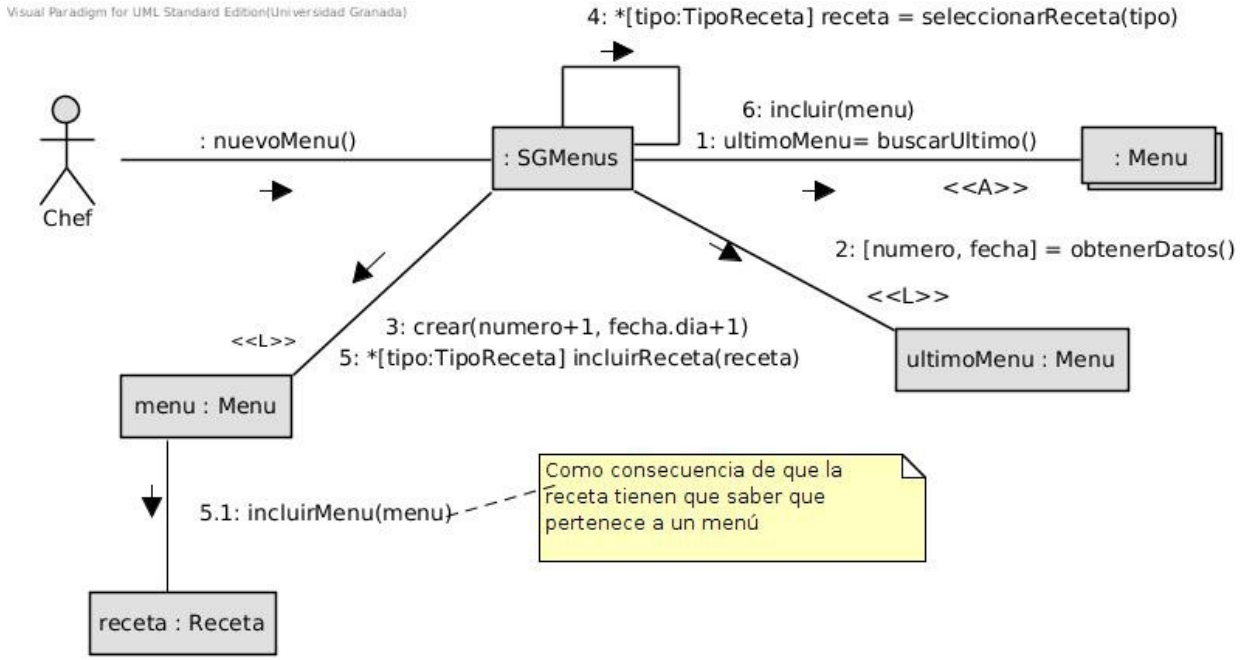


Seguimos el siguiente procedimiento:

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.

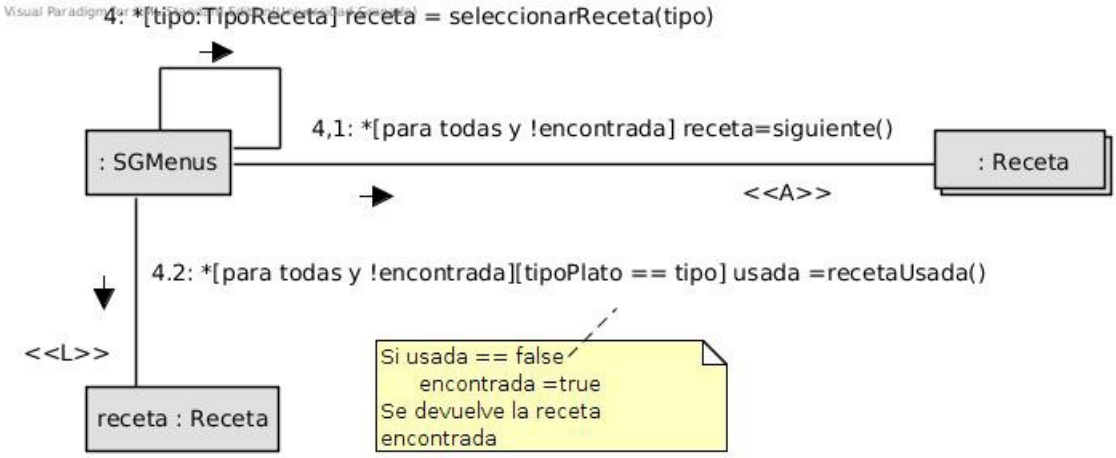
3. Identificar responsabilidades de :SGMenus
   a) Buscar el último menú, para obtener su número y fecha.
   b) Crear el nuevo menú a partir de estos datos.
   c) Seleccionar una receta para el primer plato, teniendo en cuenta lo indicado anteriormente y enlazarla con el menú.
   d) Seleccionar una receta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