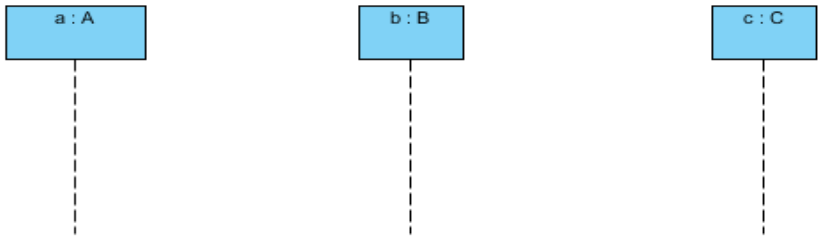
Tema 2: Lección 3

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

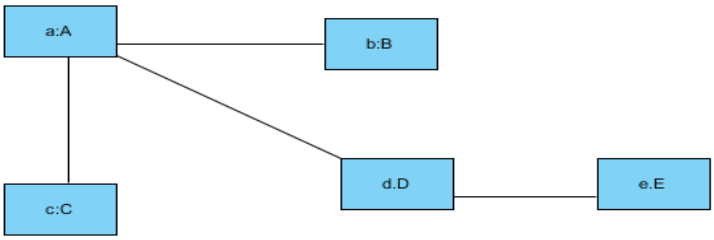
Ejercicio 2. Establece en una tabla la correspondencia entre los diagramas de secuencia y los diagramas de comunicación, indicando cómo se representan los siguientes elementos: objetos, mensajes, canal de comunicación, estructuras de control, subordinación en el envío de mensaje y orden de un mensaje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

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

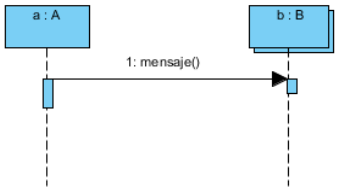
Ejercicio 4. A partir del siguiente esquema elabora un diagrama de secuencia genérico en el que haya al menos dos fragmentos combinados distintos. Tradúcelo a su correspondiente diagrama de comunicación e impleméntalo en Java y Ruby.

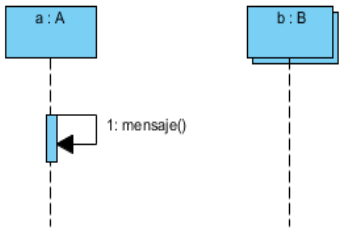


Ejercicio 5. A partir del siguiente esquema elabora un diagrama de comunicación genérico en el que haya al menos una estructura de control iterativa y otra selectiva. Tradúcelo a su correspondiente diagrama de secuencia e impleméntalo en Java y Ruby.



Ejercicio 6: Los siguientes diagramas ofrecen dos alternativas de representación sobre una operación que se realiza sobre una colección. Ambos diagramas pueden ser válidos, dependiendo de la operación concreta que se desee realizar sobre la colección: (a) incluir: añadir un elemento, (b) borrar: eliminar un elemento, (c) limpiar: eliminar todos sus elementos, (d) existe: comprobar si existe o no un elemento, (e) devolver: buscar un elemento concreto dada una posición o una clave, y (f) iterar: recorrer todos sus elementos. Justifica la equivalencia o no de los diagramas, en base a las operaciones y escribe en Java y Ruby el código correspondiente a ellos.

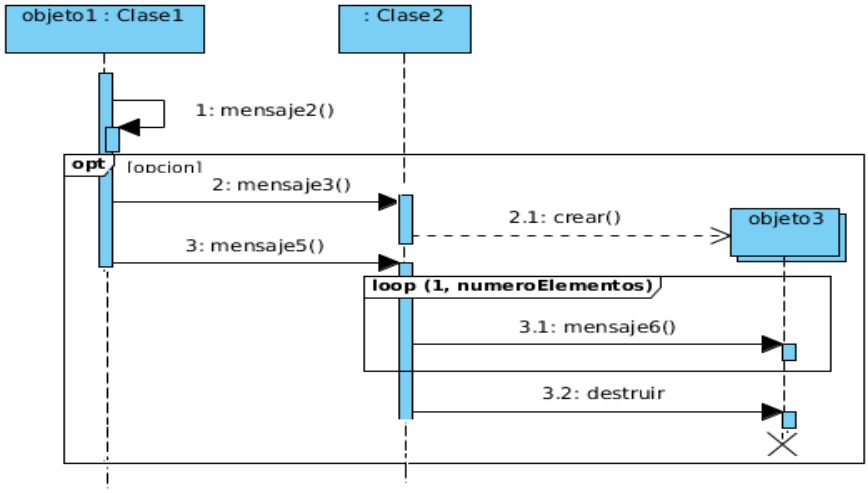




b es un atributo de a, manejado por a.

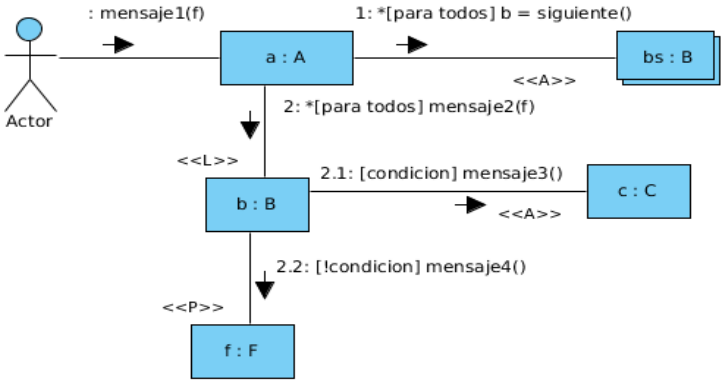
Ejercicio 7: Dado el siguiente Diagrama de secuencia, responde verdadero (V) o falso (F) a las

siguientes cuestiones



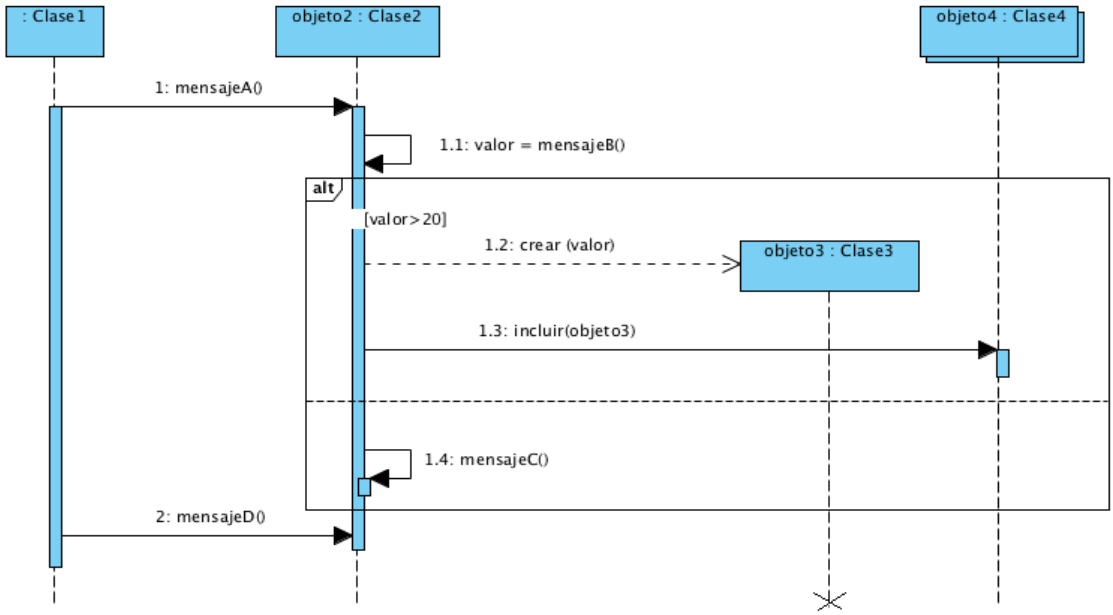
El envío de mensaje 1 (**mensaje2()**) es un envío de mensaje a **self** y además **recursivo**	
El objeto **objeto3** es un objeto que vive sólo en esta operación	
En la clase **Clase2** tienen que estar definidos los siguientes métodos: **mensaje3(), mensaje5() y mensaje6()**	
E l **fragmento combinado** 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 **es correcto**: *class Clase 2* *def mensaje5* *objeto3.each [|obj| obj.mensaje6()]* *objeto3=nil* *end* *end*	

Ejercicio 8: 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: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** {...}	
El la clase B debe estar implementado el método **siguiente()** para poder responder al envío de mensaje número 1	
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); } }* *}*	

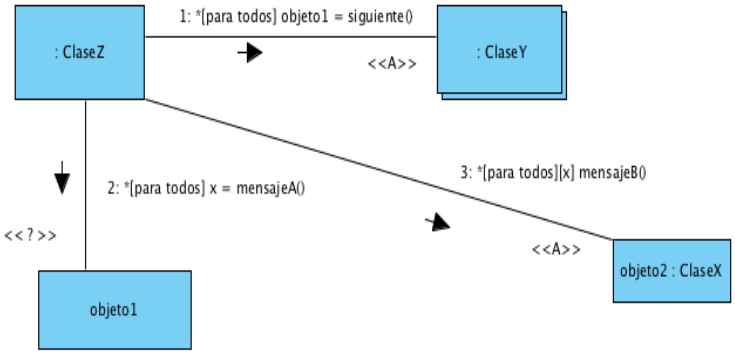
Ejercicio 9. A partir del siguiente diagrama de secuencia, responde verdadero (V) o falso (F) a las siguientes cuestiones



La clase **Clase2** debe tener implementado 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>> o <<L>>	
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 10. A partir del siguiente diagrama de comunicación, responde verdadero (V) o falso (F)

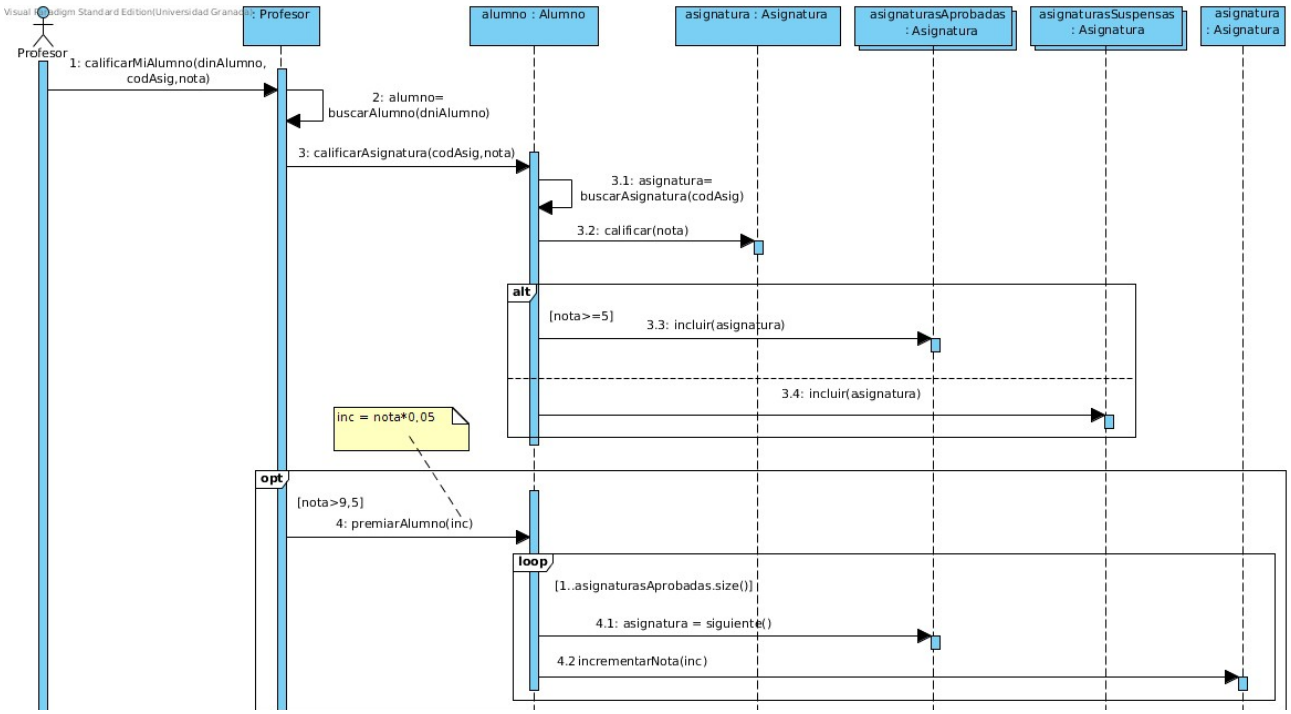
a las siguientes cuestiones



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 11. Escribe el código correspondiente en Java y Ruby para los diagramas de los ejercicios 7 al 10.

Ejercicio 12. A partir del siguiente diagrama de secuencia, responde a las siguientes cuestiones.



A) Explica en lenguaje natural cuál es el algoritmo usado por el profesor para calificar a uno de sus alumnos.

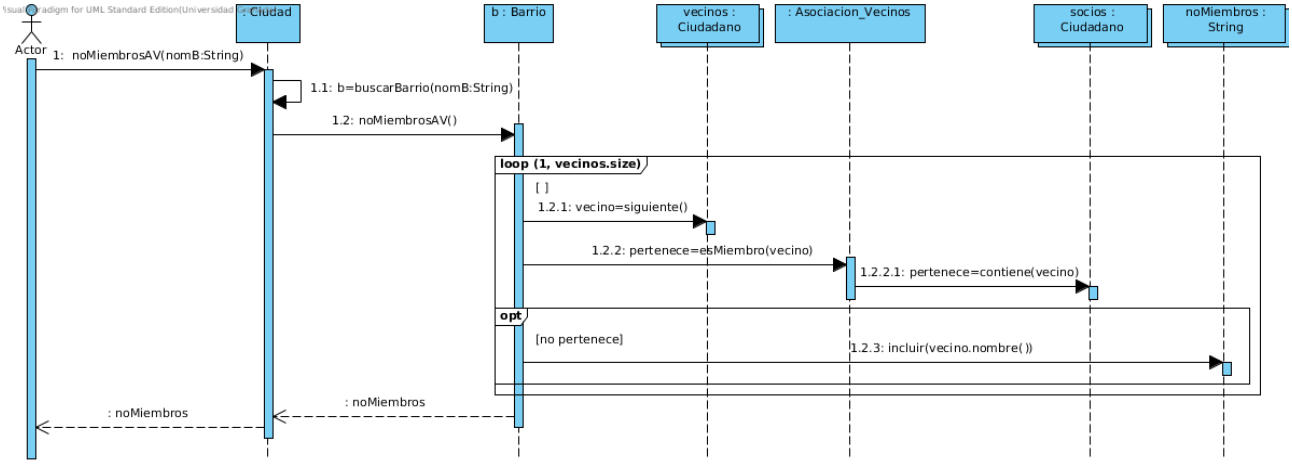
B) Tradúcelo a diagrama de comunicación.

C) Obtén el diagrama de clases que se deriva de él.

D) Implementa en Java y el Ruby el método **premiarAlumno(inc)** en la clase correspondiente

E) Cambia el diagrama para que se modifique la forma de premiar a un alumno, pasando a ser de la siguiente forma: se elige una asignatura de las suspensas que tenga nota superior a 4 y se le aprueba con un 5.

Ejercicio 13. En relación al diagrama de clases del ejercicio 9 de la relación de ejercicios 2.1, tenemos el siguiente diagrama de secuencia para la operación **noMiembrosAv** de la clase Ciudad:

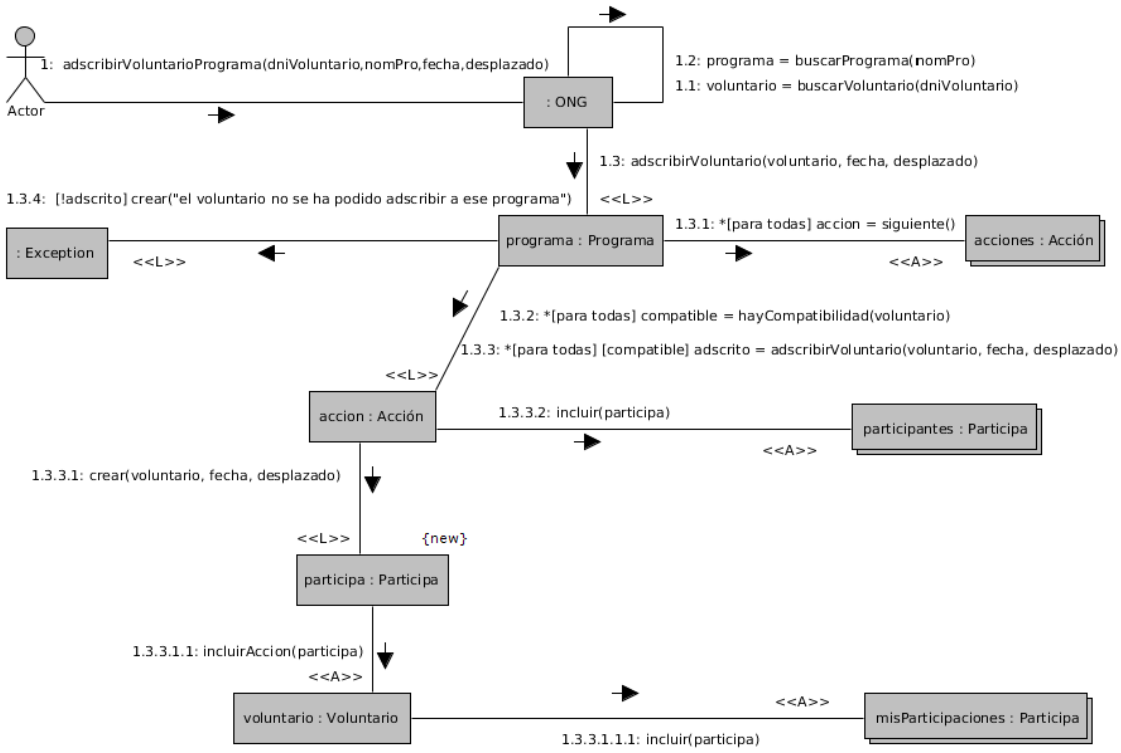


A) Comprueba qué correspondencia hay entre el diagrama de clases y el de secuencia.

B) Tradúcelo a Diagrama de comunicación

C) Implementa en Java y en Ruby todos los métodos especificados en el diagrama de secuencia.

Ejercicio 14. A partir del siguiente diagrama de comunicación, correspondiente al ejercicio 10 de la relación 2.1, para la operación AdscribirVoluntarioPrograma de la clase ONG:



A) Intenta entender qué se expresa en él y responde V(verdadero) o F(falso) a las siguientes cuestiones:

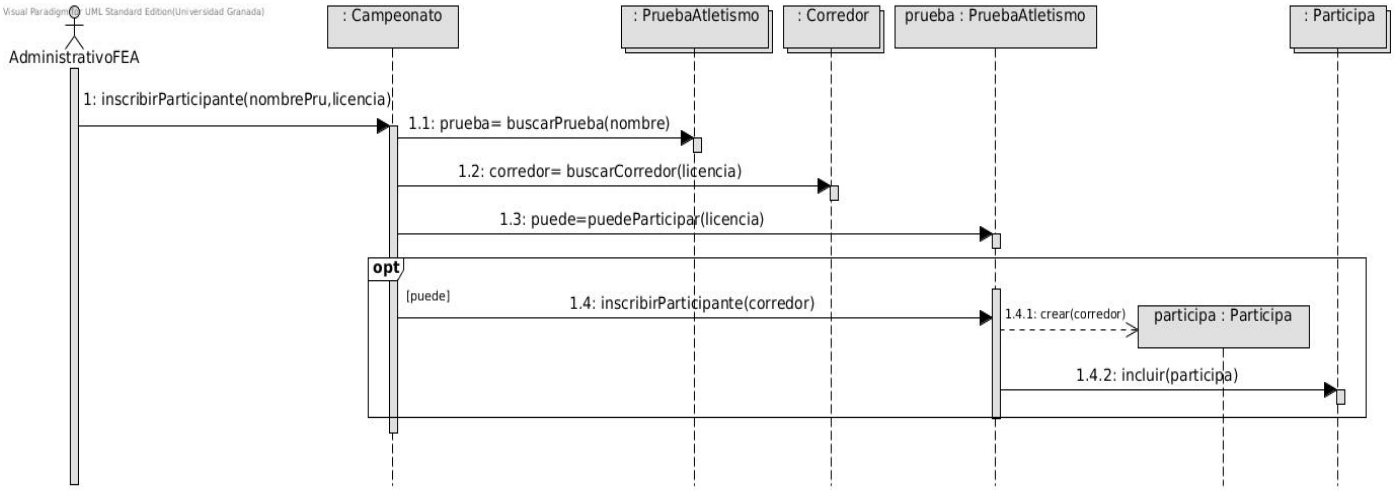
En el envío de mensaje 1.2 el objeto receptor es **self/this.**	
El envío de mensaje 1.3.1 significa que a todas las acciones del programa le vamos a adscribir un voluntario.	
En el método **crear** de la clase **Participa** (envío de mensaje 1.3.3.1 y subordinados) se construye un enlace entre el objeto p**articipa** y el objeto v**oluntario**.	
El enlace entre el objeto **Acción** y el multiobjeto de la clase **Participa** estereotipado como <<A>> significa que el objeto **accion** conoce al multiobjeto sólo 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.**

D) Indica si en el diagrama de clases asociado falta o sobra alguna operación de las que aparecen en el diagrama de secuencia

Ejercicio 15. A partir del siguiente diagrama de secuencia



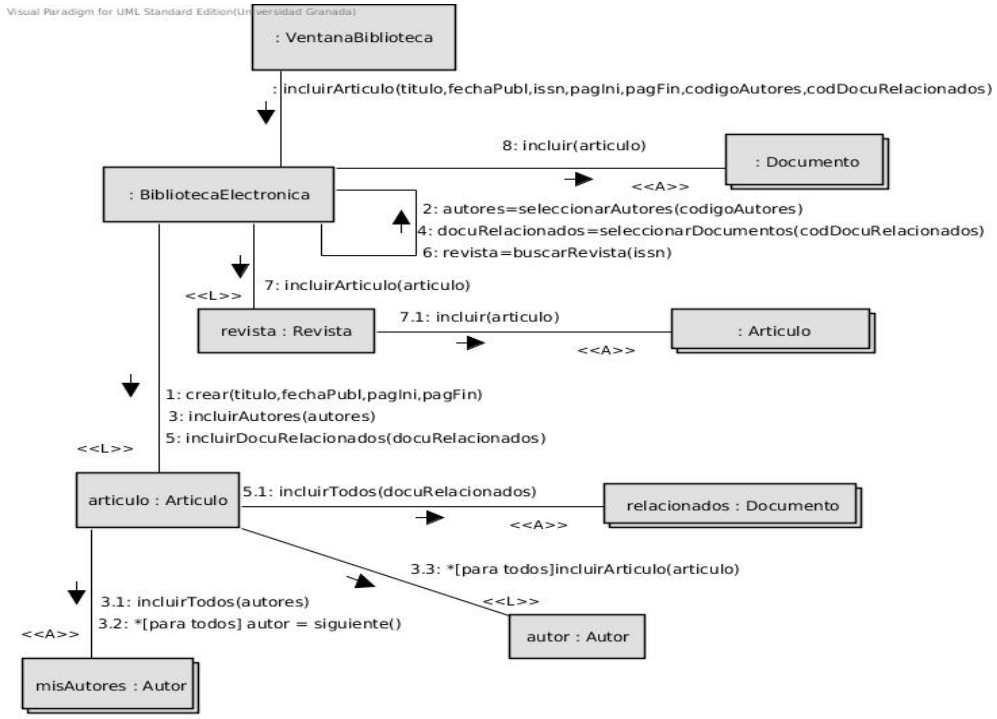
A) Responde V (verdadero) o F (Falso) a las siguientes cuestiones:

La condición del fragmento combinado **opt** está mal, tendría que estar negada	
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 1.2 es un objeto de la clase **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 16. A partir del diagrama de comunicación:



A) Intenta comprenderlo y 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 de la 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: relacionados.incluirTodos(docuRelacionados)	

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){
misAutores.addAll(autores);
for(Autor autor:misAutores)
autor.incluirArticulo(articulo);
}
}	

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

B) Implementa en Java y el Ruby el método **incluirArticulo(….)** en la clase **BibliotecaElectronica**

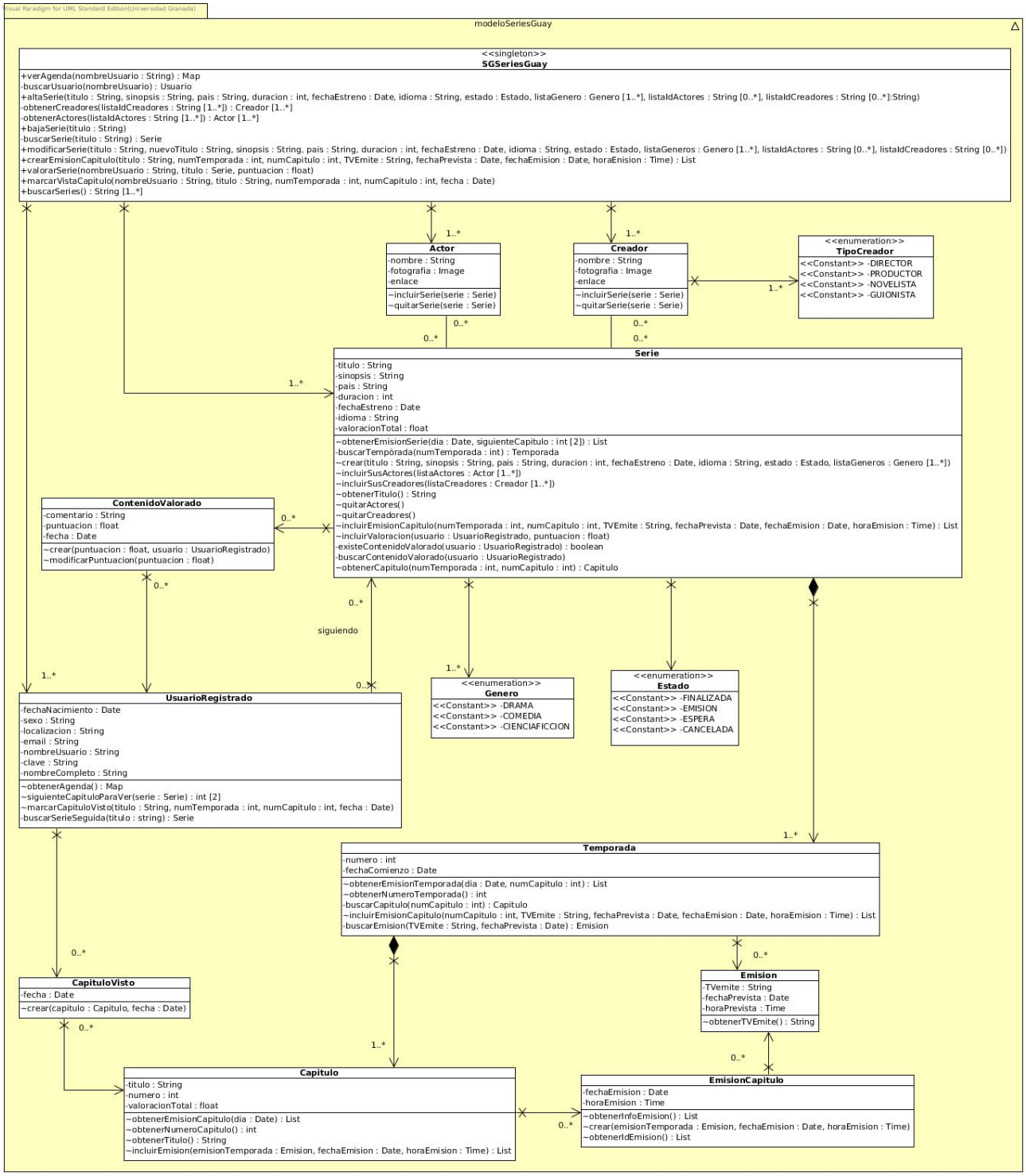
C) Obtén el Diagrama de secuencia de la operación **3:incluirAutores(autores)**

Ejercicio 17. Traduce los diagramas de la práctica 3 de secuencia a comunicación o de comunicación a secuencia según corresponda.

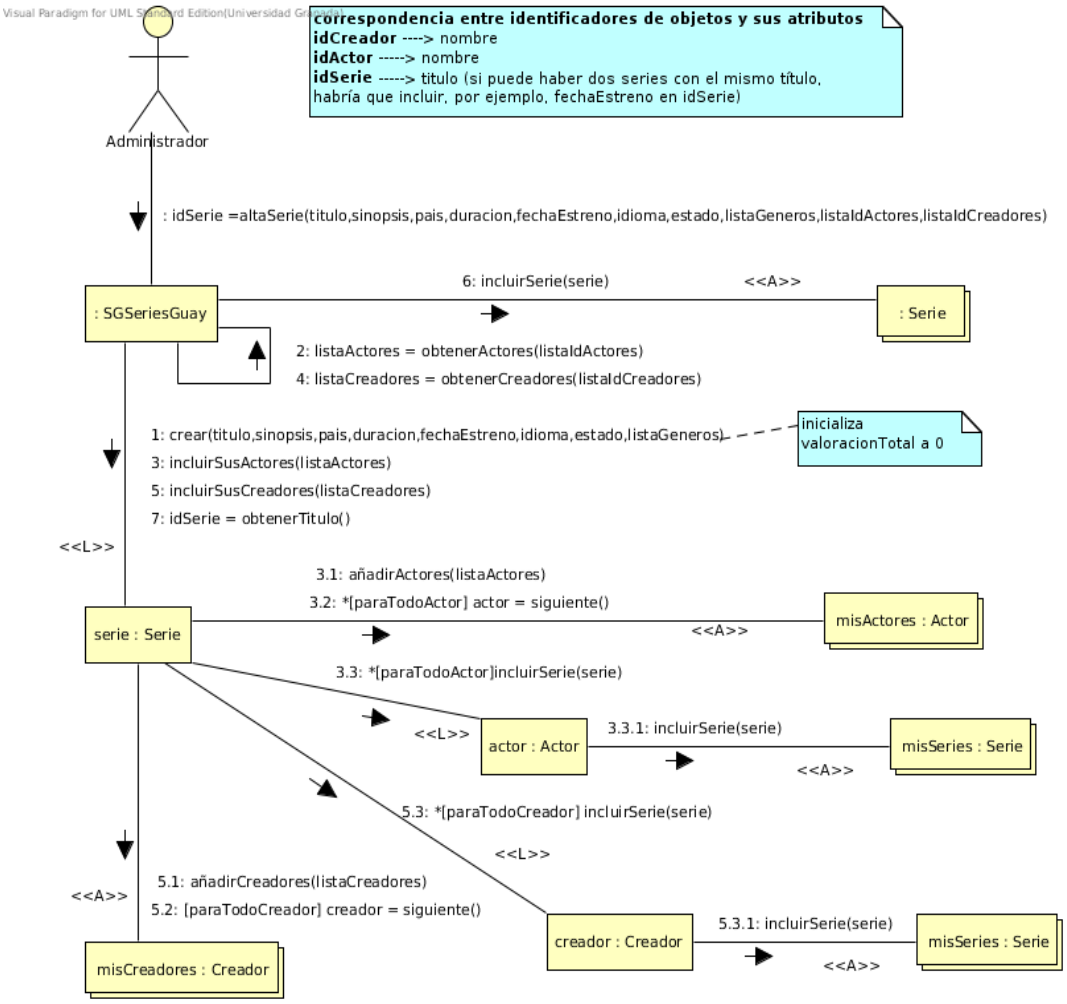
Ejercicio 18. Implementa en Java y Ruby los diagramas de comunicación que se proporcionan a continuación. Traduce al menos un diagrama de comunicación de los proporcionados a su

correspondiente diagrama de secuencia.

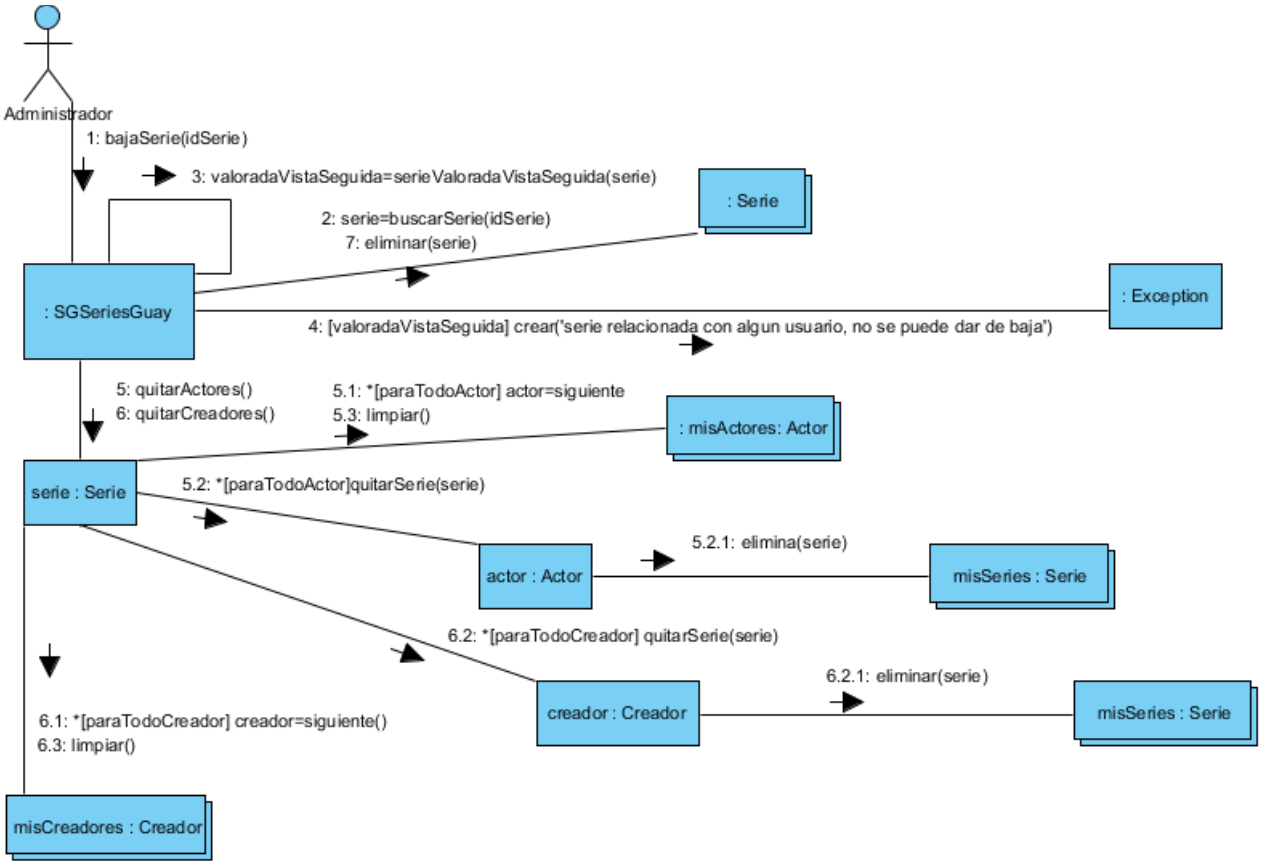
Como ayuda para la comprensión e implementación de estos diagramas se proporciona el Diagrama de Clases.



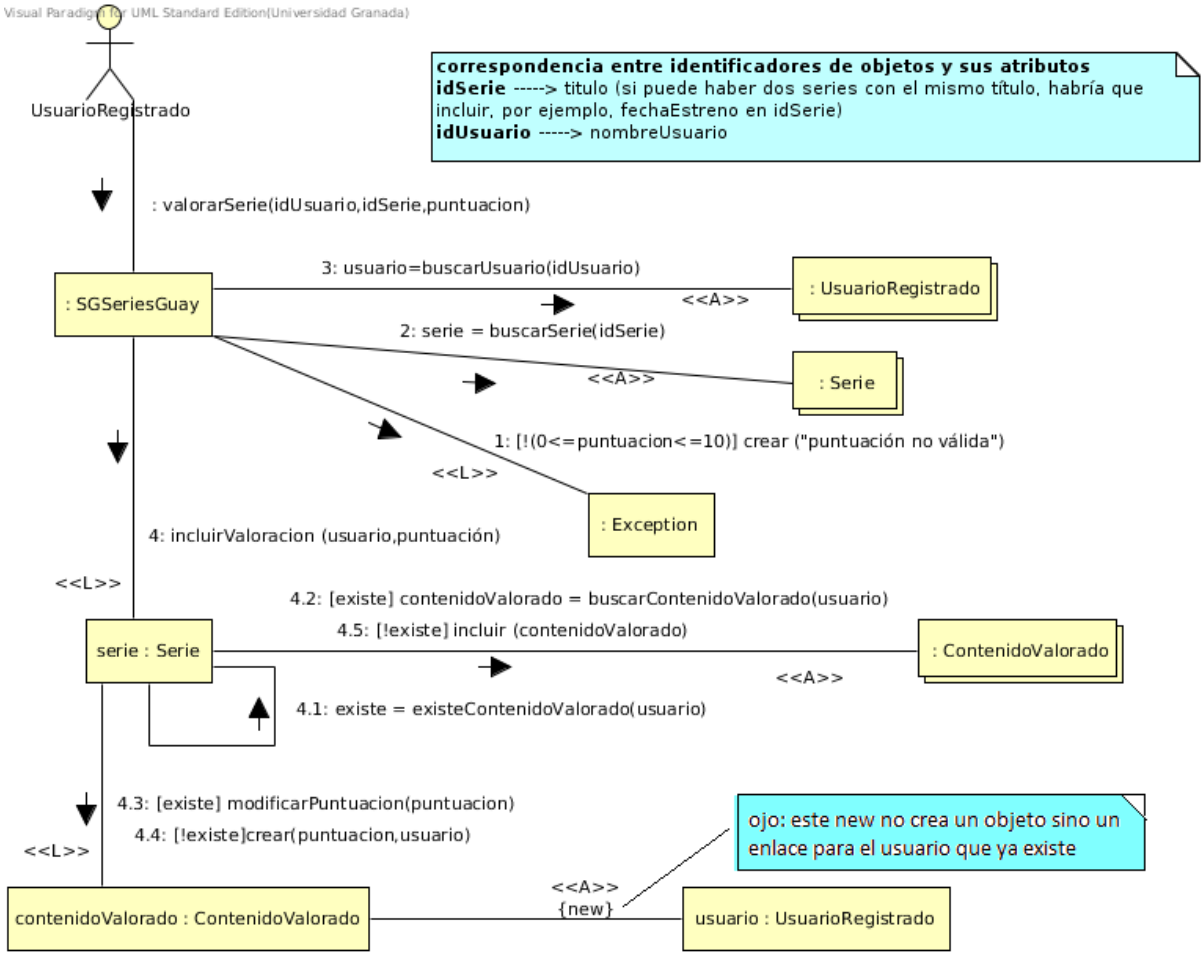
A) Incluir una nueva serie de TV en el sistema :SGSeriesGuay


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
misActores : Actor
<<A>>
3.3: *[paraTodoActor]incluirSerie(serie)
<<L>>
actor : Actor
3.3.1: incluirSerie(serie)
misSeries : Serie
<<A>>
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)
misSeries : Serie
<<A>>

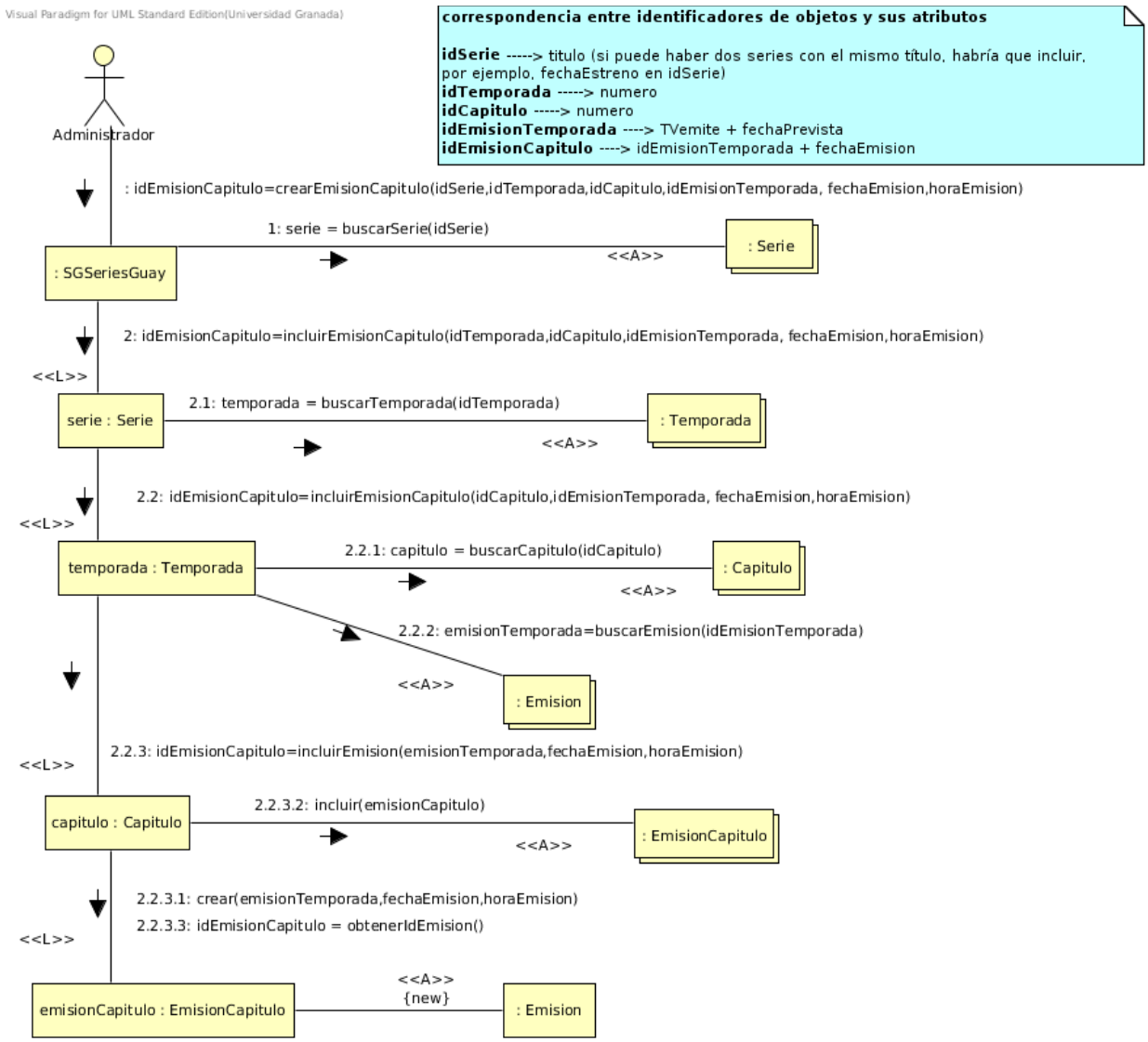
B) Eliminar una serie



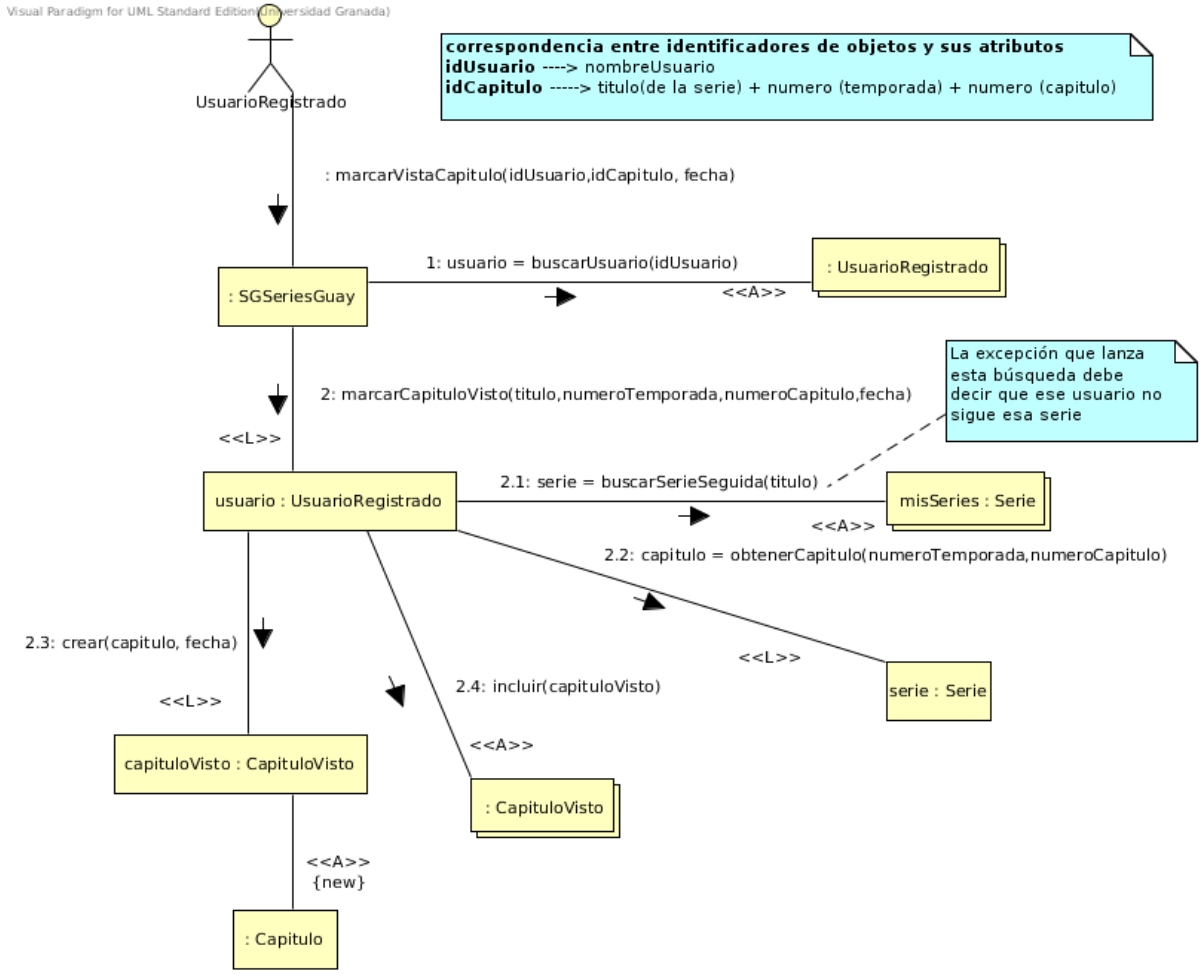
C) Valorar una serie por parte de un un usuario registrado en el sistema. Solo se valora, no se comenta.



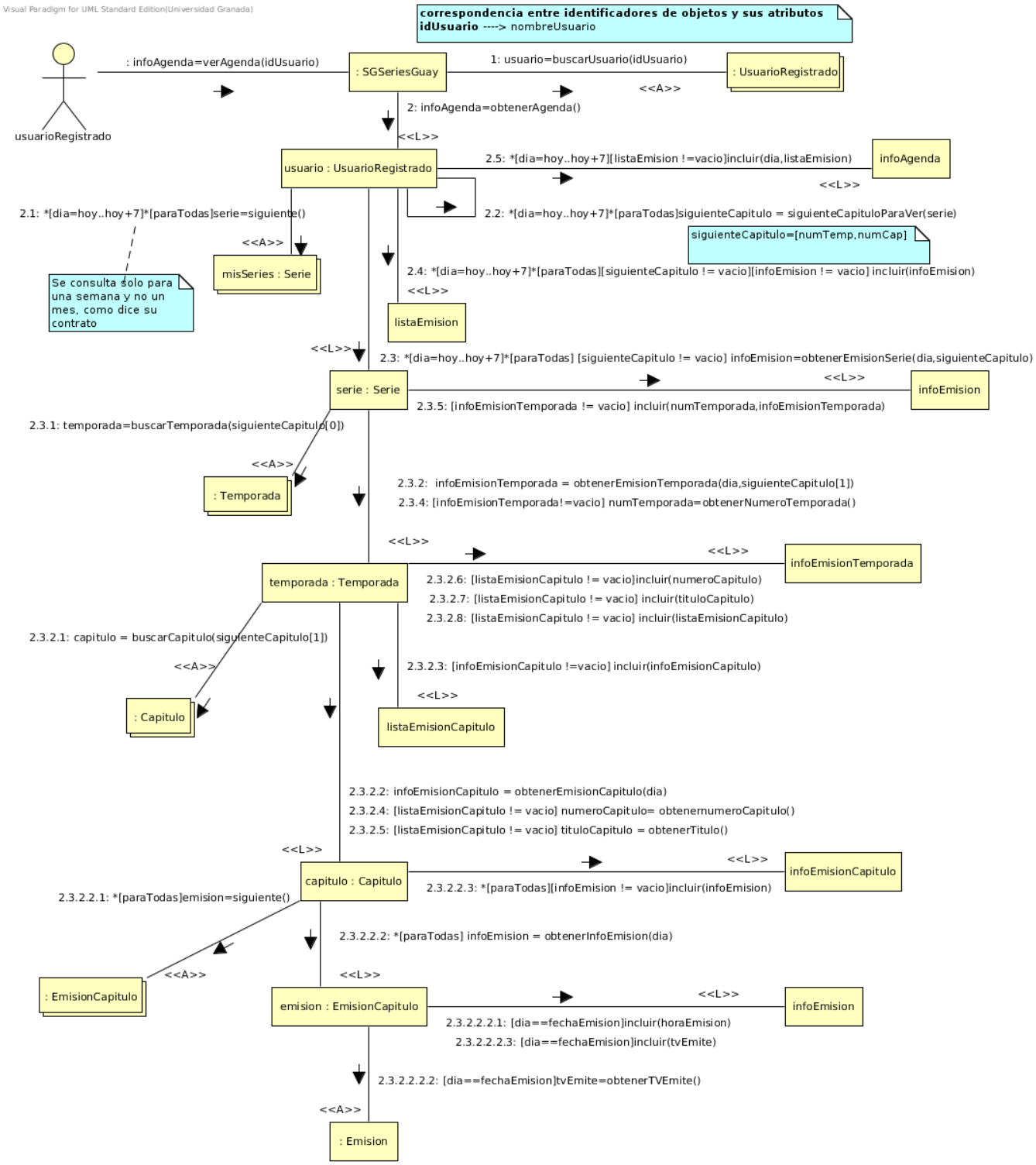
D) Definir la emisión de un capítulo de una serie



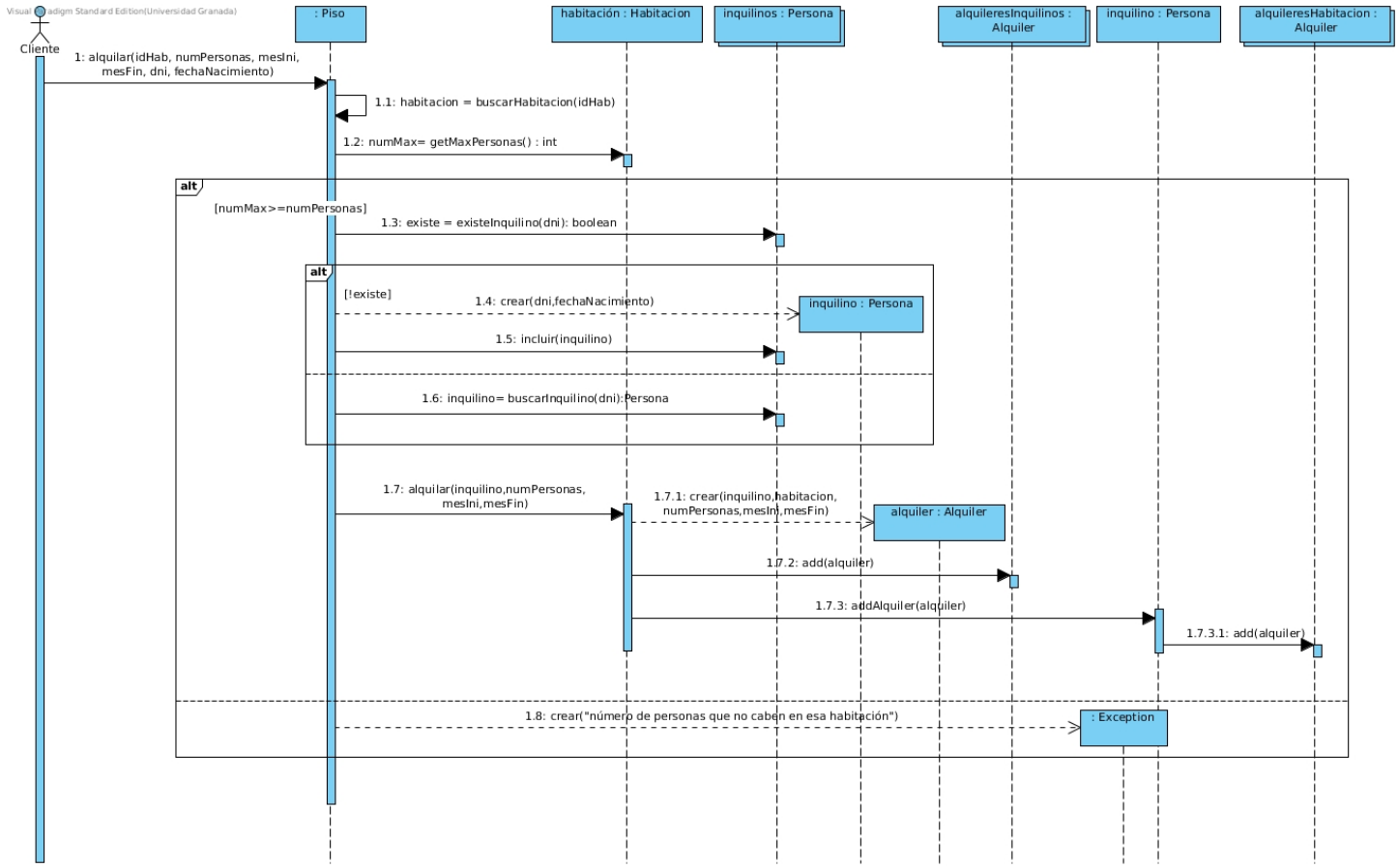
E) Marcar un capítulo como visto por un usuario registrado



F) Consultar la agenda semanal de un usuario registrado, para ver qué capítulos se emiten de las series que está siguiendo. Hay varias salidas: infoAgenda, infoEmision, infoEmisionTemporada, infoEmisionCapitulo. Todas ellas son listas heterogéneas de objetos.



Ejercicio 19. A partir del diagrama de secuencia:



A) Responde V (verdadero) o F (falso)

inquilinos es un multiobjeto	
En la clase **Alquiler** debe añadirse el método **add(alquiler:Alquiler)**	
El mensaje **alquilar(...)** al objeto de la clase **Habitacion** no se puede llamarse así, pues ya existe otro al objeto de la clase **Piso** con el mismo nombre	
En envío de mensaje 1.7.3 entre el objeto **habitación** y el objeto **inquilino** puede enviarse debido a que existe un canal de comunicación entre ellos de tipo <<P>>	
El mensaje 1.1 se envía de forma recursiva	
Los envíos de mensaje 1.4 y 1.8 están mal, debe ser una línea continua	

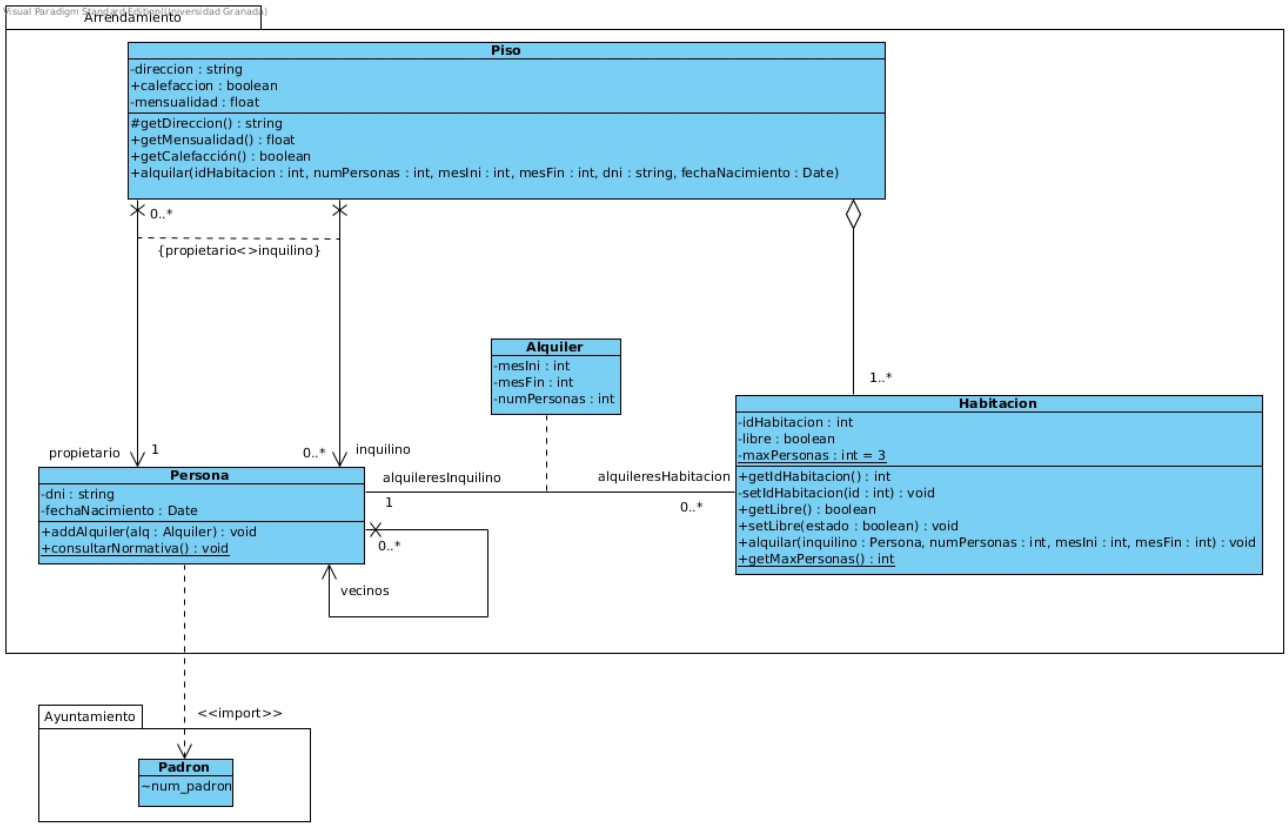
B) Trata de entender cómo se alquila una habitación en el piso e implémentalo en Java y el Ruby.

C) Cambia el fragmento combinado de tipo *alt* más externo por un fragmento combinado de tipo *break*, de tal forma que en el fragmento de interacción del break se incluya la excepción.

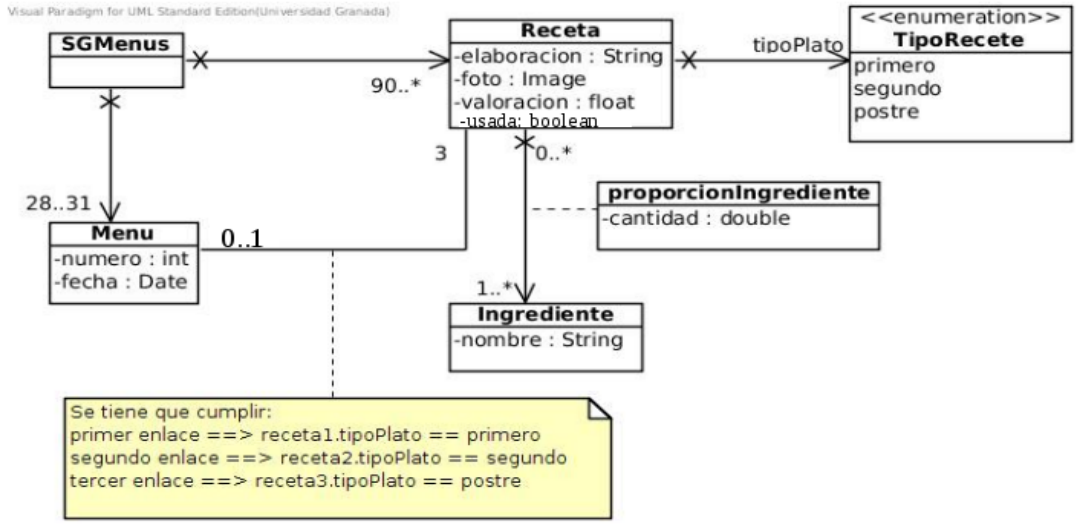
D) Modifica el diagrama para que se compruebe si la habitación está libre o no antes de alquilarla, si está libre se puede alquilar si no lo está se lanzará una excepción.

E) En el diagrama de secuencia proporcionado solo una persona puede alquilar una habitación, con la limitación de que el numero de personas para las que se alquila la habitación quepan ella. El dueño del piso quiere cambiar la forma de alquilar las habitaciones, una habitación puede ser alquilada por más de una persona siempre que queden suficientes plazas libres en ella.

F) El diagrama de clases de la siguiente página representa la estructura de clases del ejercicio. Modifícalo para que incluya todo lo que se contempla en el diagrama de secuencia proporcionado. ¿Qué habría que cambiar para que estuviese conforme con la modificación indicada en el apartado E?



Ejercicio 20 (RESUELTO). Partiendo del siguiente diagrama de clases, que se corresponde con una de las posibles soluciones al ejercicio 14.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 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 Menú existe un método obtenerDatos() que devuelve el número de menú y la fecha.

Ejercicio 21. Partiendo de los diagramas de clases obtenidos en la relación de problemas del tema 2.2 en el ejercicio 14 de los supuestos B a G, obtener el diagrama de interacción (secuencia o comunicación) de las siguientes operaciones

A) (del 14.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, indicar 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 14.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 14.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.

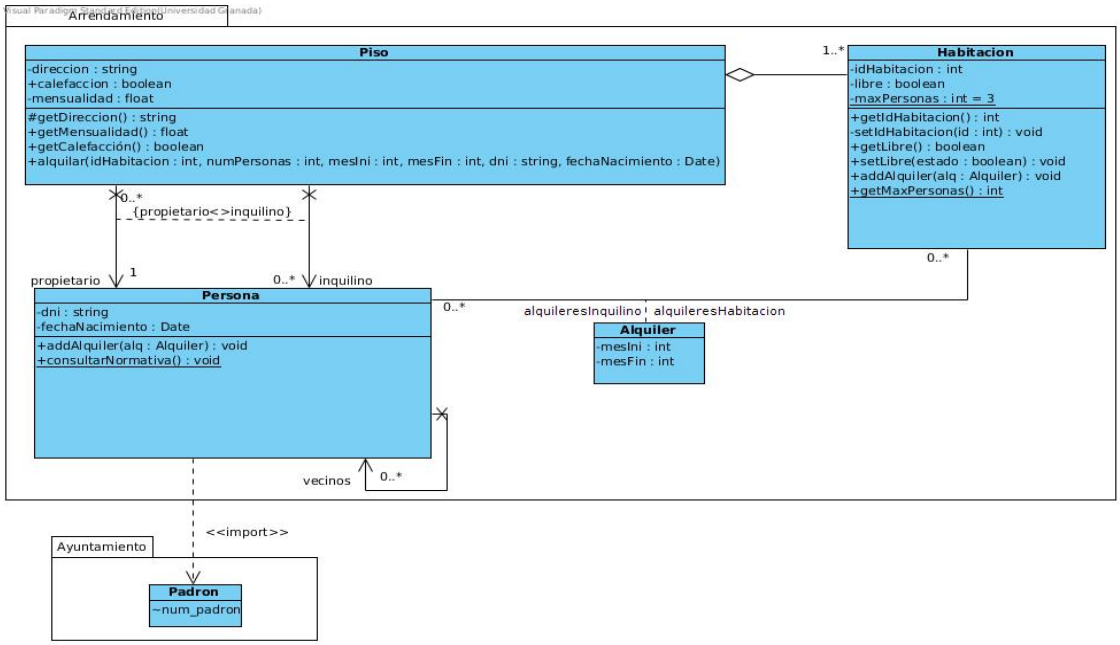
D) (del 14.E) Programar un evento asignándole una sala en la que se va a desarrollar. Para ello hay que proporcionar el nombre del evento y las fechas en las que se va a celebrar, a partir de esas fechas hay que buscar una sala libre. Una vez terminada la operación se ha creado un objeto evento y se 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.

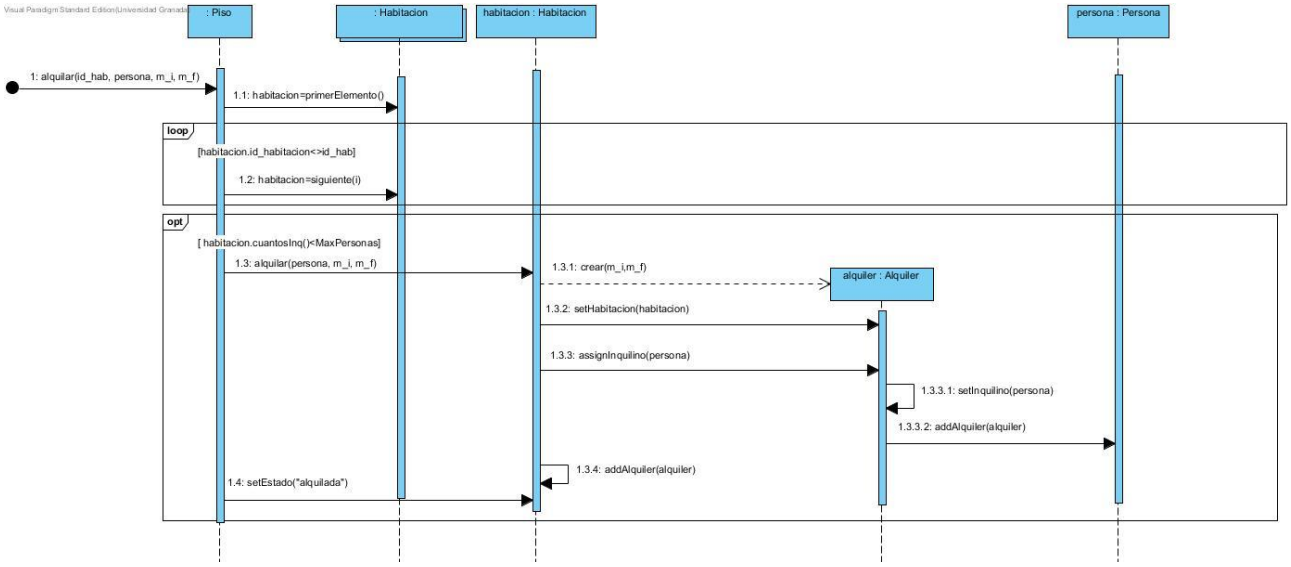
E) (del 14.F) Incluir un paquete en una ruta. Para ello hay que indicar el código del paquete a incluir la parada en la que hay que recoger y entregar el paquete y el peso del paquete, teniendo en cuenta que en ningún momento se debe superar el peso máximo de la furgoneta, por lo que habrá que calcular cual es el peso de la furgoneta en todas las paradas por las que tiene que pasar el paquete, si en algunas de ellas sobrepasa el peso máximo permitido, el paquete no podrá enviarse en esa ruta.

F) (del 14.G) 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.

G) (del 14.H) Incluir un nuevo polígono proporcionando sus vértices y comprobando que es un polígono regular, si no lo es, se debe lanzar una excepción y no permitir su construcción. Una vez terminada la operación se ha creado un objeto polígono regular y tantos objetos punto como vértices se hayan proporcionado y enlazado en el polígono con sus vértices.

Ejercicio 22. Partiendo de los siguientes diagrama de clases y diagrama de secuencia, responde a las preguntas que hay a continuación.





A) Preguntas sobre el diagrama de clases. Responde verdadero (V) o falso (F). Cada respuesta incorrecta resta una correcta.

La asociación **Piso-Habitación** es de agregación	
{propietario<>inquilino} es una restricción	
Una **persona** puede saber de qué **pisos** es **propietaria**	
Alquiler es una clase asociación	
vecinos es un rol	
Puede haber una **habitación** sin **inquilinos**	

Piso tiene un atributo de referencia de la clase **Persona**	
El método **getDireccion** de **Piso** es método de clase, no de instancia	
Desde la clase **Piso** se puede acceder directamente al atributo **dni** de un inquilino	
Desde la clase **Persona** se puede acceder directamente al atributo **num_padron** de un objeto	
de la clase **Padron**	
attr_accessor en Ruby hace que los atributos sean públicos	
Los siguientes objetos de **Piso** tienen la misma identidad: *Piso piso1= new Piso("Calle Alta", true, 600)* *Piso piso2= new Piso("Calle Alta", true, 600)*	

B) Teniendo en cuenta sólo la información presentada en el diagrama de clases, implementa en Java la clase **Persona** (deja en blanco el contenido del método **consultarNormativa**). Añade su constructor para inicializar los atributos y métodos consultores para todos los atributos. No te olvides de incluir también el código necesario para indicar que cada clase está dentro de un paquete o módulo, y si precisa algo de otro paquete o módulo.

C) Teniendo en cuenta sólo la información presentada en el diagrama de clases, implementa en Ruby la clase **Alquiler**. Añade su constructor para inicializar los atributos y métodos consultores para todos los atributos. No te olvides de incluir también el código necesario para indicar que cada clase está dentro de un paquete o módulo, y si precisa algo de otro paquete o módulo.

D) Preguntas sobre el diagrama de secuencia. Responde verdadero (V) o falso (F).
Cada respuesta incorrecta resta una correcta.

Existe un multiobjeto anónimo	
En la clase **Habitacion** debe añadirse el método **primerElemento***()*	
El método **assignInquilino(Persona persona)** debe estar implementado en la clase **Habitacion**	
En un Diagrama de Comunicación equivalente, el canal de comunicación entre **alquiler:Alquiler** y **persona:Persona** *y* estaría estereotipado con **<<A>>**	
El mensaje 1.3.4 se envía de forma recursiva	
***[habitacion.id_habitacion<>id_hab]** debe estar delante del mensaje *siguiente* en un Diagrama de Comunicación equivalente en lugar del fragmento combinado **loop**	

E) Implementa en Ruby el diagrama de secuencia del método **alquilar** de la clase **Habitacion**.

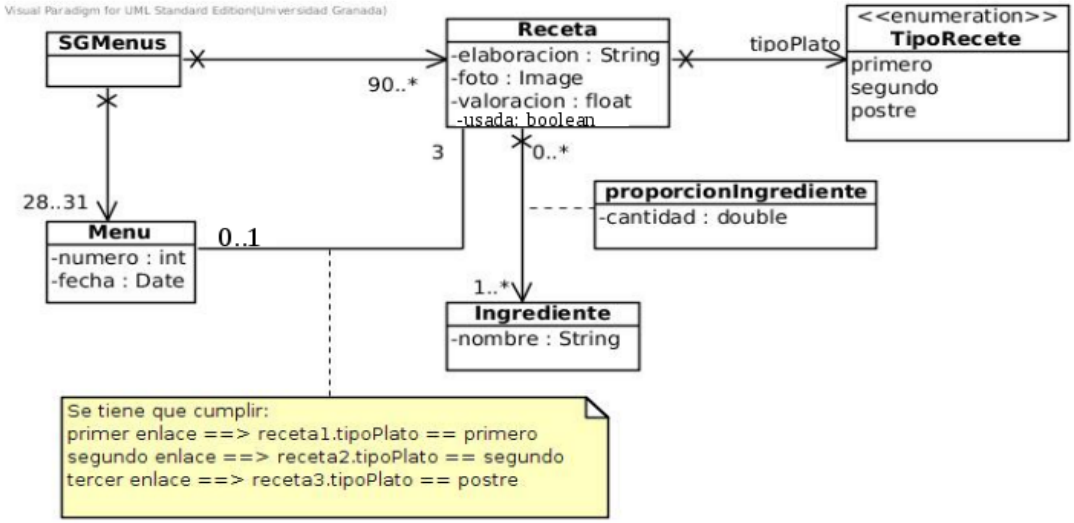
F) Implementa en Ruby los métodos necesarios en las demás clases para completar el diagrama de secuencia del método **alquilar** de la clase **Habitacion**.

Relación de problemas (PDOO) **Curso: 2016-17**

Solución de ejercicios

Ejercicio 20.

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



Seguimos el siguiente procedimiento:

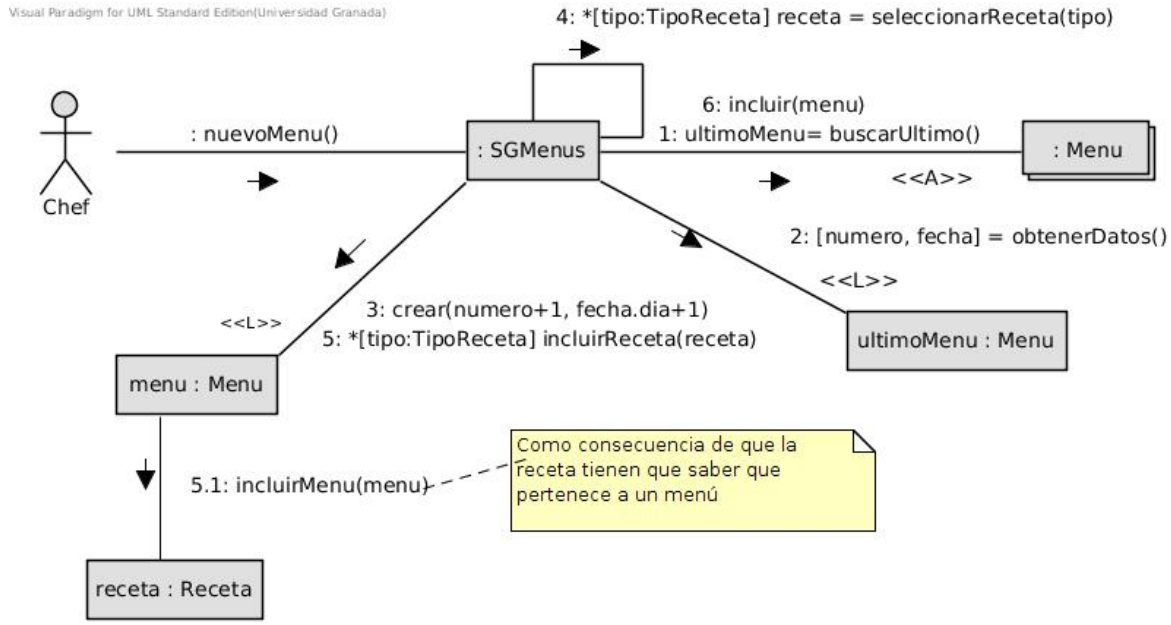
1. Identificar los parámetros de la operación: Operación que no necesita ningún parámetro ya que todo lo que necesita está incluido en el sistema. La operación quedaría:
   nuevoMenu()

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. (1) (2)
   b) Crear el nuevo menú a partir de estos datos. (3)
   c) Seleccionar una receta para el primer plato y enlazarla con el menú. (4) (5)
   d) Seleccionar una receta para el segundo plato y enlazarla con el menú. (4) (5)
   e) Seleccionar una receta para el plato postre y enlazarla con el menú. (4) (5)
   f) Indicar a la receta que pertenece a un menú. (5.1)
   g) Incluir el nuevo menú dentro de la lista de menús.(6)

Entre paréntesis el número de mensaje correspondiente en el siguiente diagrama.

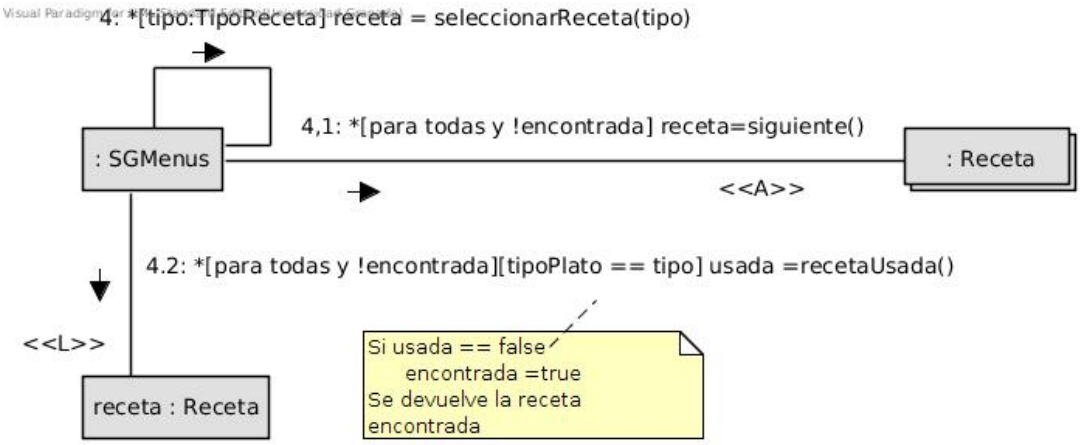
4. Dibujar el diagrama con el primer nivel de subordinación:



5. Ampliar el diagrama con el segundo nivel de subordinación, en este caso para el envío de mensaje **seleccionarReceta()** y responsabilidad de **:SGMenus**

   - Responsabilidad de **:SGMenus**

     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 válida



6. Ejercicios relacionados con el problema resuelto:

   1. Refina el modelo obtenido, encontrando otras posibles soluciones.
   2. Pásalo a diagrama de secuencia el diagrama de comunicación obtenido.

3. Implémentalo en Java y el Ruby

Ejercicio 22.

A)

La asociación **Piso-Habitación** es de agregación	V
{propietario<>inquilino} es una restricción	V
Una **persona** puede saber de qué **pisos** es **propietaria**	F
Alquiler es una clase asociación	V
vecinos es un rol	V
Puede haber una **habitación** sin **inquilinos**	V
Piso tiene un atributo de referencia de la clase **Persona**	V
El método **getDireccion** de **Piso** es método de clase, no de instancia	F
Desde la clase **Piso** se puede acceder directamente al atributo **dni** de un inquilino	F
Desde la clase **Persona** se puede acceder directamente al atributo **num_padron** de un objeto	F
de la clase **Padron**	
attr_accessor en Ruby hace que los atributos sean públicos	F
Los siguientes objetos de **Piso** tienen la misma identidad: *Piso piso1= new Piso("Calle Alta", true, 600)* *Piso piso2= new Piso("Calle Alta", true, 600)*	F

B)

```
package Arrendamiento;
import Ayuntamiento.Padron; // para la clase Padron
import java.util.Date; // para la clase Date
import java.util.ArrayList; // para la clase ArrayList

public class Persona {
        private String dni;
        private Date fechaNacimiento;
        private ArrayList<Persona> vecinos; // Relación involutiva
        private ArrayList<Alquiler> alquileresInquilino; // Lista alquileres
// constructor (opción 1)
        public Persona (String unDni, Date fecha) {
                this.dni = unDni;
                this.fechaNacimiento = fecha;
                this.vecinos = new ArrayList();
                this.alquileresInquilino = new ArrayList();
        }
// constructor (opción 2)
        public Persona (String unDni, Date fecha, ArrayList<Persona> vecs,
                ArrayList<Alquier> alqs) {
                this.dni = unDni;
```

```
                this.fechaNacimiento = fecha;
                this.vecinos = vecs;
                this.alquileresInquilino = alqs;
        }
// Método de clase
        public static void consultarNormativa(){...}
// Añadir un alquiler nuevo
        public void addAlquiler (Alquiler alq) {...}
// Getters
        public String getDNI(){ return this.dni; }
        public Date getFechaNacimiento(){ return this.fechaNacimiento; }
        public ArrayList<Persona> getvecinos() { return this.vecinos.get(i); }
// Setters (Aunque no se piden en el enunciado podrían ser estos)
        public void setDNI(String unDNI){ this.dni = unDNI; }
        public void setFechaNacimiento(Date unaFecha){
                this.fechaNacimiento = unaFecha; }
        public void setvecinos(ArrayList<Persona> vecs){ this.vecinos=vecs; }
}
```

C)

```
module Arrendamiento
class Alquiler

attr_reader :mesIni, :mesFin, :inquilino, :habitacion

def initialize (mesIni, mesFin, inqui, habit)
        @mesIni = mesIni
        @mesFin = mesFin
        @inquilino = inquil
        @habitacion = habit
end
end
```

D)

Existe un multiobjeto anónimo	V
En la clase **Habitacion** debe añadirse el método **primerElemento***()*	F
El método **assignInquilino(Persona persona)** debe estar implementado en la clase **Habitacion**	F
En un Diagrama de Comunicación equivalente, el canal de comunicación entre **alquiler:Alquiler** y **persona:Persona** *y* estaría estereotipado con **<<A>>**	V
El mensaje 1.3.4 se envía de forma recursiva	F
***[habitacion.id_habitacion<>id_hab]** debe estar delante del mensaje *siguiente* en un Diagrama de Comunicación equivalente en lugar del fragmento combinado **loop**	V

E)

```
class Habitacion
# Sólo definimos los métodos utilizados en el diagrama, los demás (como el constructor)
# suponemos que ya están
public
```

```
    def alquilar(persona, m_i, m_f)
            alquiler = Alquiler.new(m_i,m_f) # 1.3.1
            alquiler.setHabitacion(self) # 1.3.2
            alquiler.assignInquilino(persona) # 1.3.3
            addAlquiler(alquiler) # 1.3.4
    end
    def addAlquiler (alq)
    # no escribimos nada para este método (no sabemos su flujo interno)
    end
    end

    F)

    class Alquiler
            def initialize (mesIni, mesFin)
                    @mesIni = mesIni
                    @mesFin = mesFin
            end
            def assignInquilino (person)
                    setInquilino(person) # 1.3.3.1
                    person.addAlquiler(self) # 1.3.3.2
            end
    # Setters
            def setHabitacion (habit)
                    # implementación del método (no sabemos su flujo interno)
            end
            def setInquilino (inquil)
                    # implementación del método (no sabemos su flujo interno)
            end
    end

    class Persona
    attr_accessor :alquiler
            # suponemos implementado el constructor de Persona
            public
            def addAlquiler (alq)
                    # implementación del método (no sabemos su flujo interno)
            end
end
```