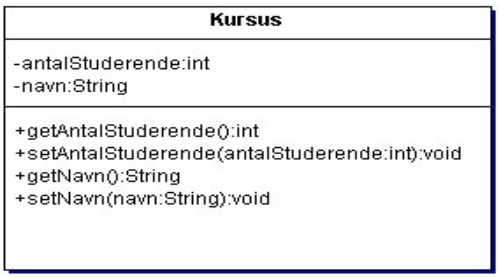
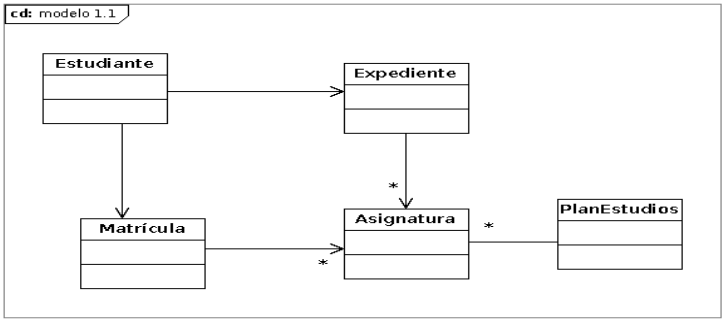
Tema 2: Lección 2

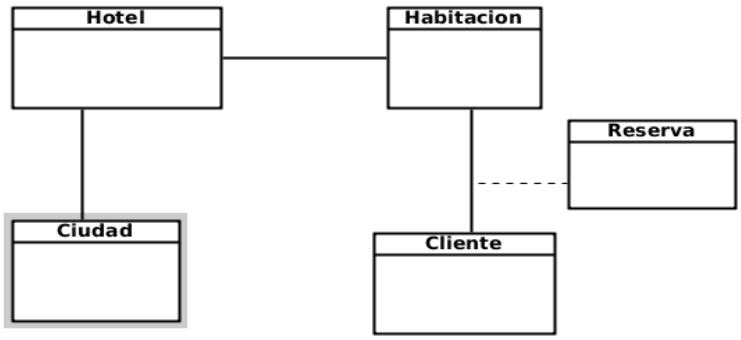
Ejercicio 1. UML es un lenguaje “universal”. Escribe el código Java y Ruby correspondiente a la declaración de la siguiente clase en danés (incluyendo la declaración de atributos y la cabecera de los métodos).



Ejercicio 2. 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.

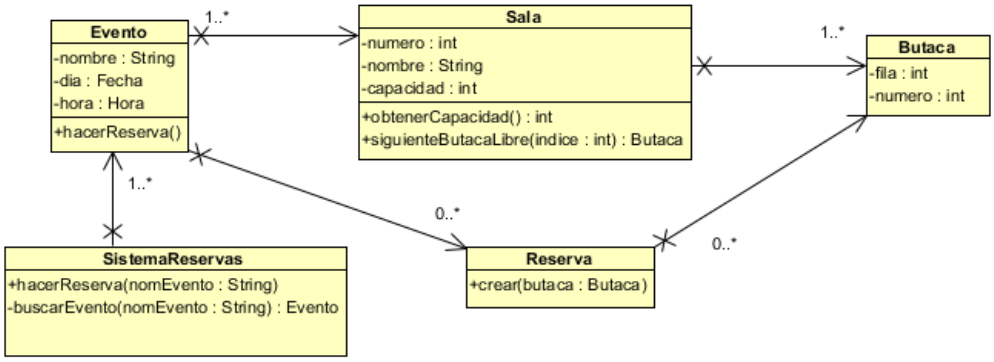


Ejercicio 3. El siguiente diagrama de clases representa hoteles con sus habitaciones y las reservas hechas por sus clientes:



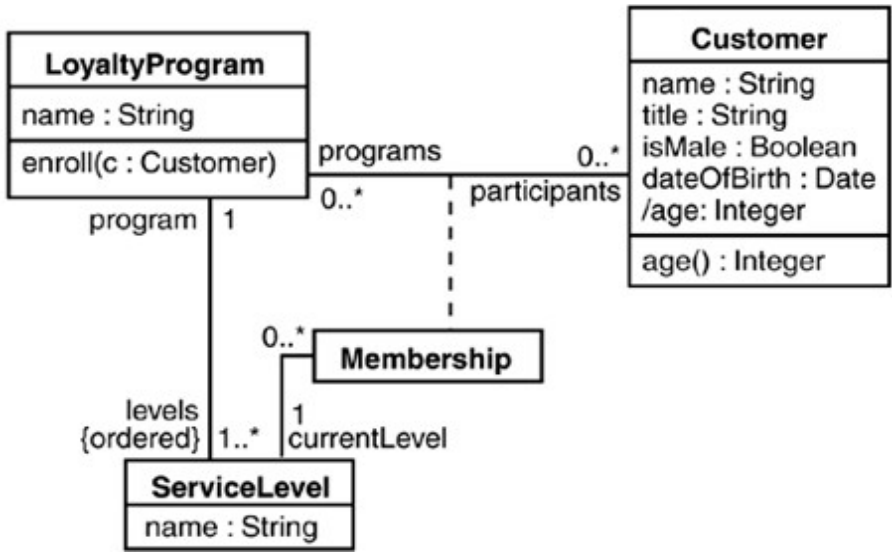
A) Complétalo con los atributos de las clases y el nombre, roles, multiplicidad y navegabilidad de las asociaciones, haciendo las suposiciones que consideres oportunas.
B) Implementa en Java el resultado obtenido.
C) Implementa en Ruby el resultado obtenido.

Ejercicio 4. A partir del siguiente diagrama de clases.



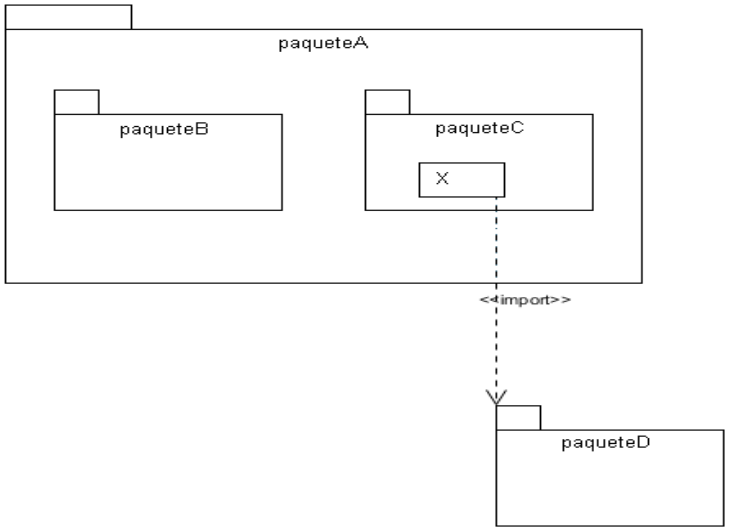
A) Indica el nombre o roles de las asociaciones.
B) ¿La asociación que existe entre Sala y Butaca podría ser una agregación?¿Y una composición?
C) Partiendo de una sala, ¿podría saberse para qué eventos está siendo usada?
D) Escribe el código Java correspondiente.
E) Escribe el código Ruby correspondiente.

Ejercicio 5. 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 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 6. Dado el siguiente diagrama de paquetes:



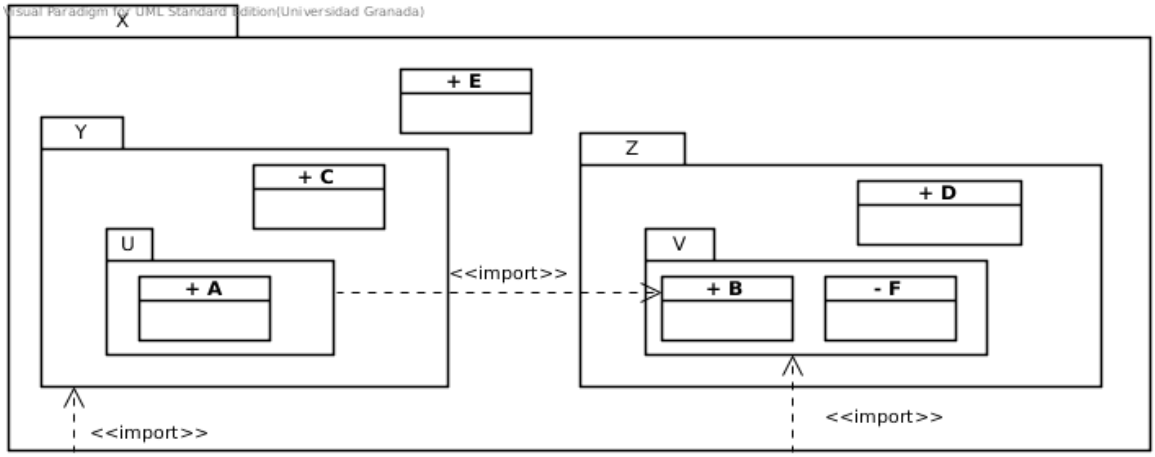
A) Indica cómo implementarías esta estructura en Java y en Ruby

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

C) Responde a los mismos casos anteriores pero considerando visibilidad pública.

Ejercicio 7. Dado el siguiente diagrama de paquetes



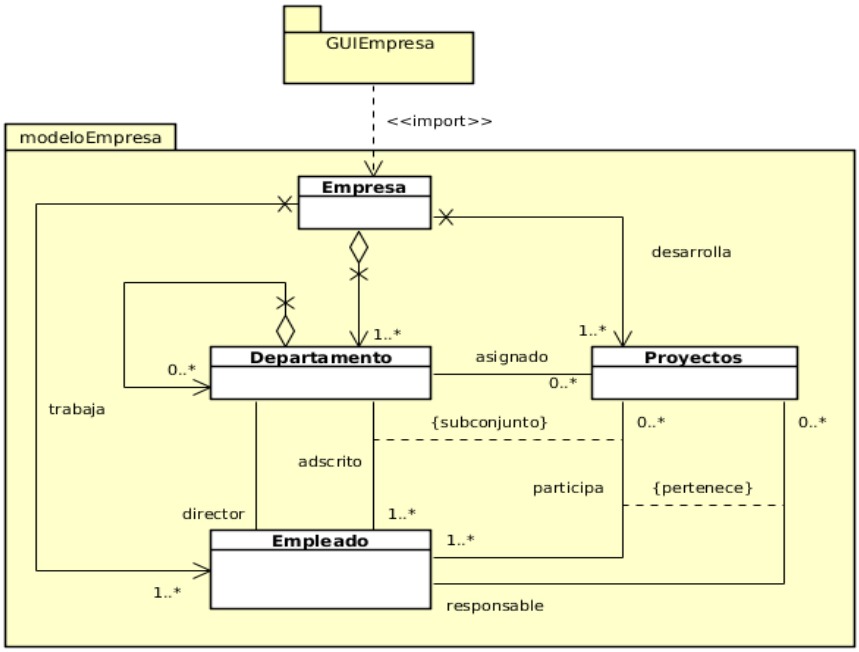
A) Indica cómo implementarías esta estructura en Java y en Ruby

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

- Desde la clase F
- Desde la clase D
- Desde la clase A

C) Si en Ruby implementamos los paquetes como módulos y cada clase está implementada en un fichero llamado <nombre_clase>.rb su módulo correspondiente: ¿Qué tendríamos que escribir en el fichero de la clase C, para poder usar un método definido como público en la clase D desde la clase C?

Ejercicio 8. A partir del siguiente diagrama de clases, responde a las cuestiones plateadas:



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 Proyectos?

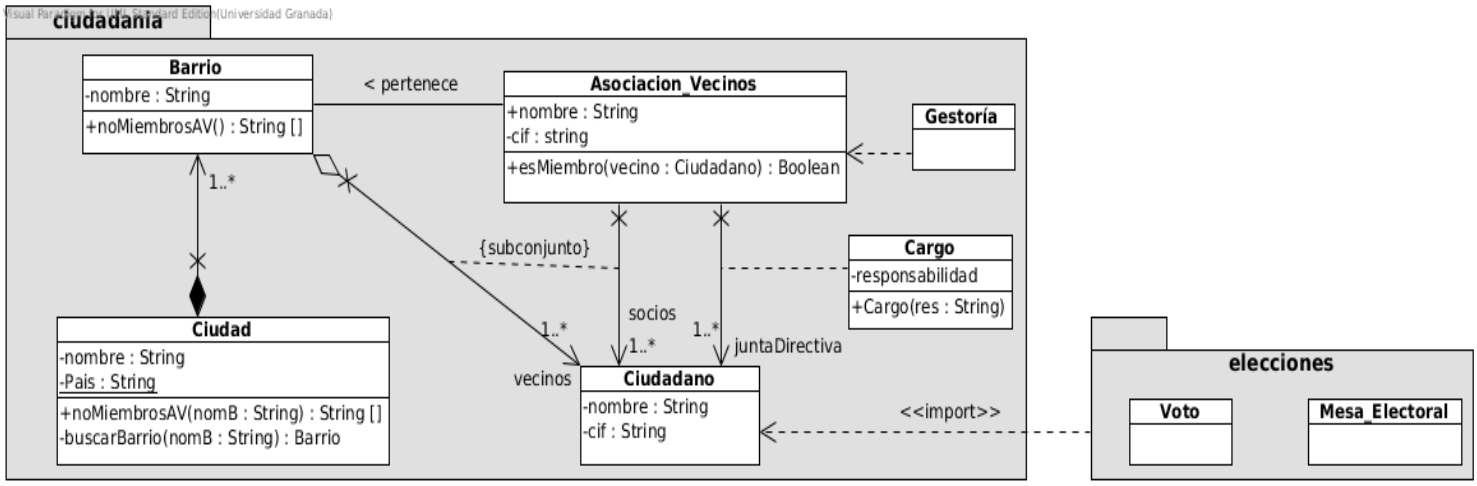
G) ¿Qué representa la línea discontinua entre las asociaciones adscrito y participa y cuál es su significado en este diagrama?

H) Hay dos asociaciones entre Proyectos y Empleado, ¿qué representan?

I) ¿Por qué hay 0..* y no 1..* en la multiplicidad de la asociación entre Empleado y Proyectos?

Relación de problemas (PDOO) **Curso: 2017-18**

Ejercicio 9 (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?

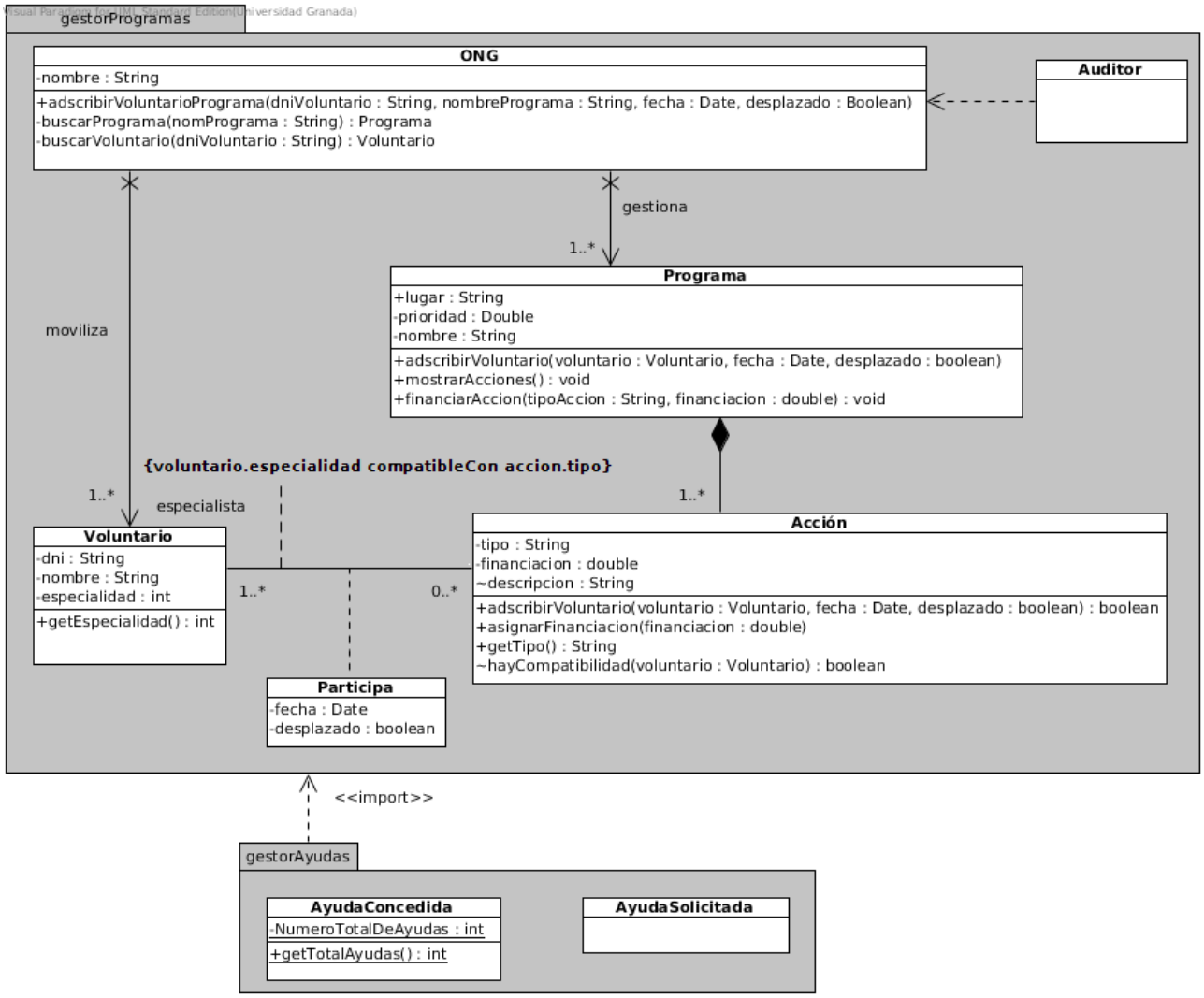
K) Define en Java los atributos de referencia de la clase Barrio.

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).

Relación de problemas (PDOO) **Curso: 2017-18**

Ejercicio 10 (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.	

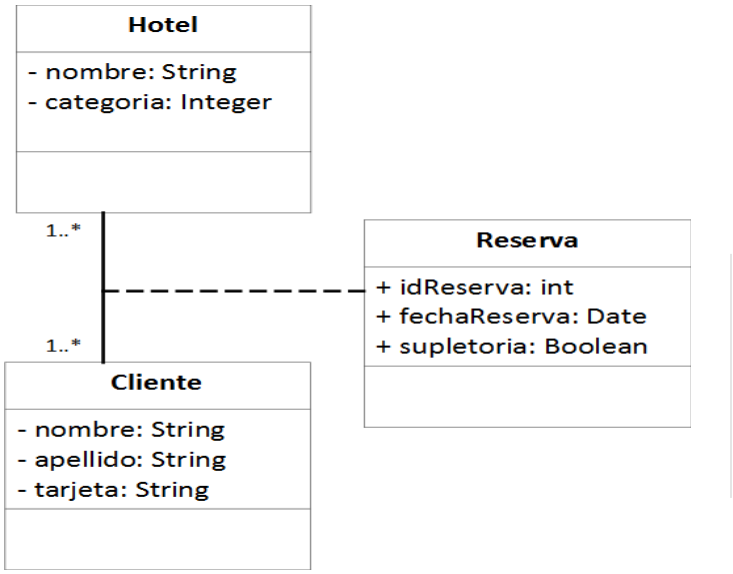
Ejercicio 11 (RESUELTO / Examen 13/14). Partiendo del diagrama de clases del ejercicio 10 responde a las siguientes preguntas:

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

Ejercicio 13. (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 14. 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 Menú 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.

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) La empresa de transporte Furespo dispone de una flota de furgonetas para distribución de paquetes por toda la península ibérica (España y Portugal). A las furgonetas se les asignan rutas. Cada ruta tiene un origen y un destino y también puede contener paradas intermedias. Las paradas se definen indicando su ciudad y distrito postal. Las rutas y sus envíos de paquetes las definen los administrativos de la empresa. Los conductores van informando a la empresa de las paradas recorridas y de los paquetes que entregan y recogen en cada una. Los administrativos pueden añadir envíos de paquetes sobre la marcha, siempre que las paradas correspondientes no hayan sido aún recorridas por la furgoneta. Cada furgoneta tiene un peso máximo de carga permitido. La carga total de las furgonetas van cambiando a lo largo de la ruta, en función de las cargas y descargas efectuadas. Los administrativos deberán tenerlo en cuenta y hacerse responsables de que la carga total de las furgonetas no sobrepase nunca su máximo permitido Un paquete se puede encontrar en tres estados según dónde se encuentre: NOCARGADO (En su parada de origen, esperando a que la furgoneta lo recoja), CARGADO (en la furgoneta) y DESCARGADO (descargado por la furgoneta en su parada de destino)

G) 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:

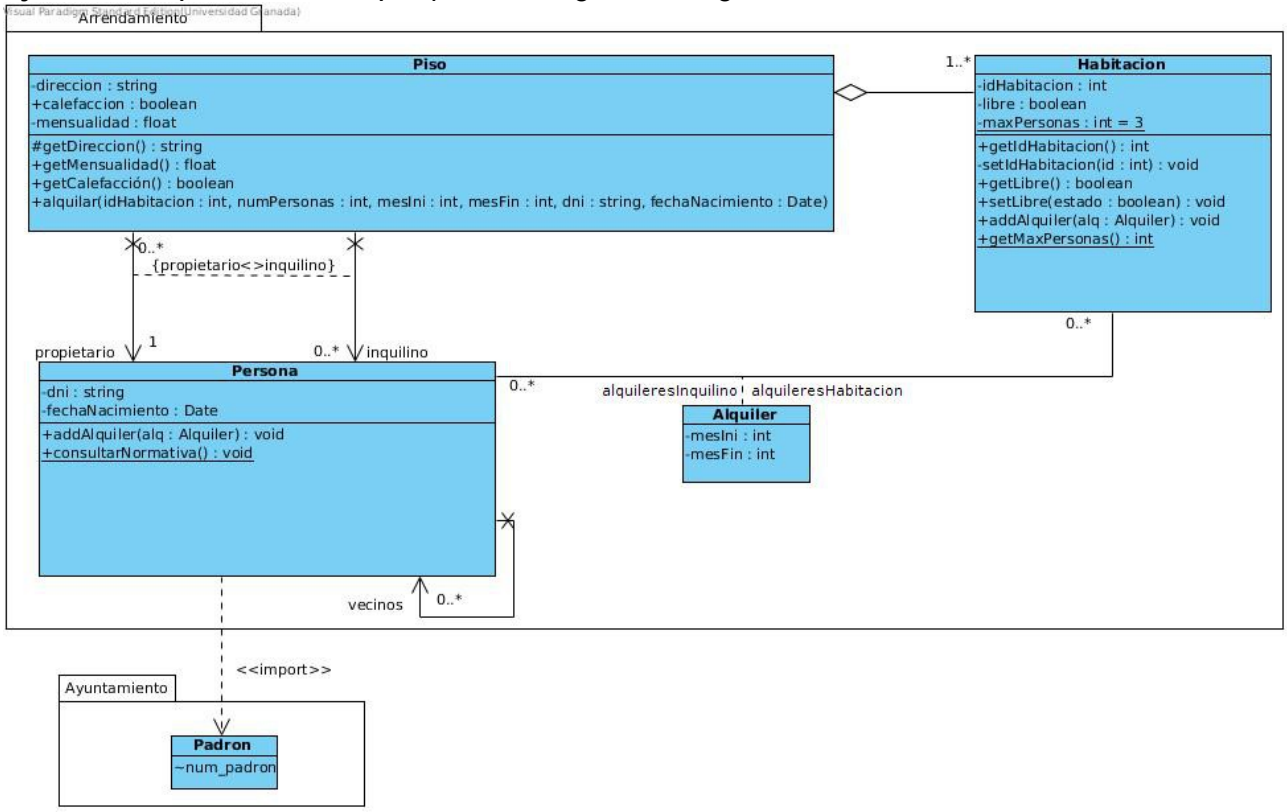
- Sólo se puede mover un disco cada vez.
- Un disco de mayor tamaño no puede descansar sobre uno más pequeño que él mismo.
- Sólo es posible desplazar el disco de arriba de la varilla

H) 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 exterior: 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 15 (Examen 15/16). A partir del siguiente diagrama de clases,



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

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*	
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)*	

Ejercicio 16. Escribe el código necesario para hacer que piso1 y piso2 sean idénticos.

Ejercicio 17. ¿Desde la clase Padrón se puede acceder al método getCalefacción de Piso? Justifica tu respuesta.

Ejercicio 18. Indica si es correcto implementar en Ruby para la clase Padrón lo siguiente:

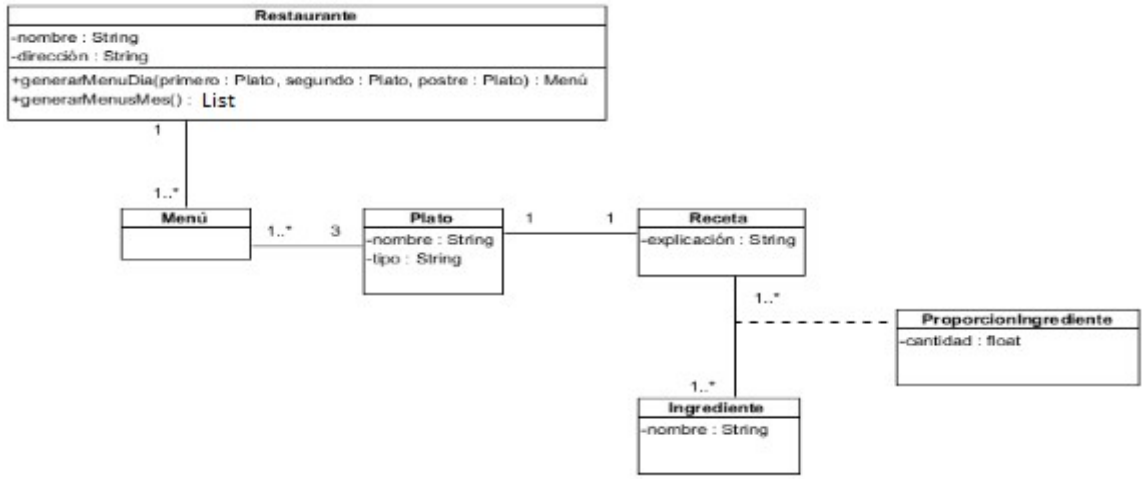
a) *attr_reader :dirección, :mensualidad*
b) *attr_accessor :calefaccion*

Justifica tu respuesta.

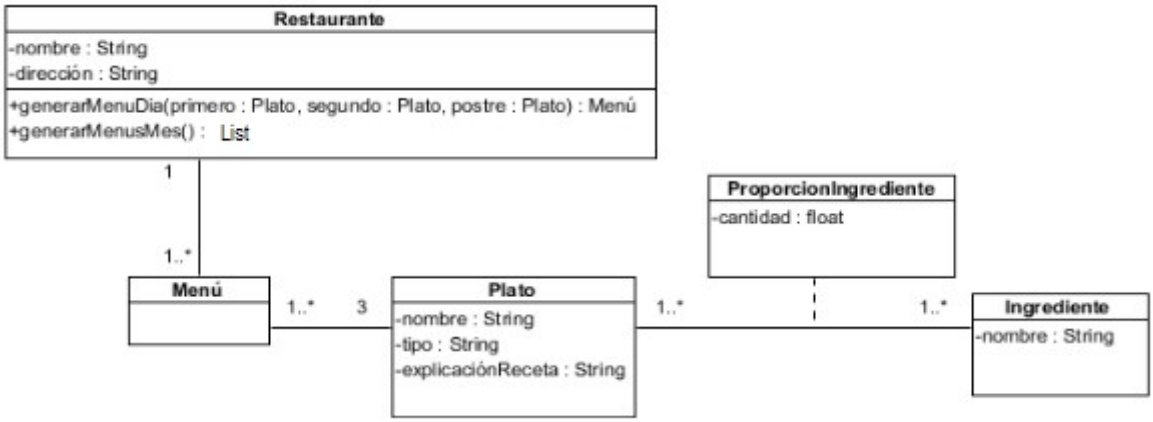
Ejercicio 19. Teniendo en cuenta sólo la información presentada en el diagrama de clases del ejercicio 15, implementa en Java y Ruby las clases *Persona, Alquiler y Padrón*. Añade sus constructores para inicializar los atributos y métodos consultores para todos los atributos. No te olvides de incluir 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. Supón que en Ruby cada clase se implementa dentro de un fichero con el mismo nombre que la clase y que están todas en la misma carpeta. Dado que no tenemos información sobre los métodos de Persona, implementa solo sus cabeceras.

Ejercicio 20. A continuación se muestran varios diseños, todos congruentes con la especificación del ejercicio 14.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 14.A en la comparativa

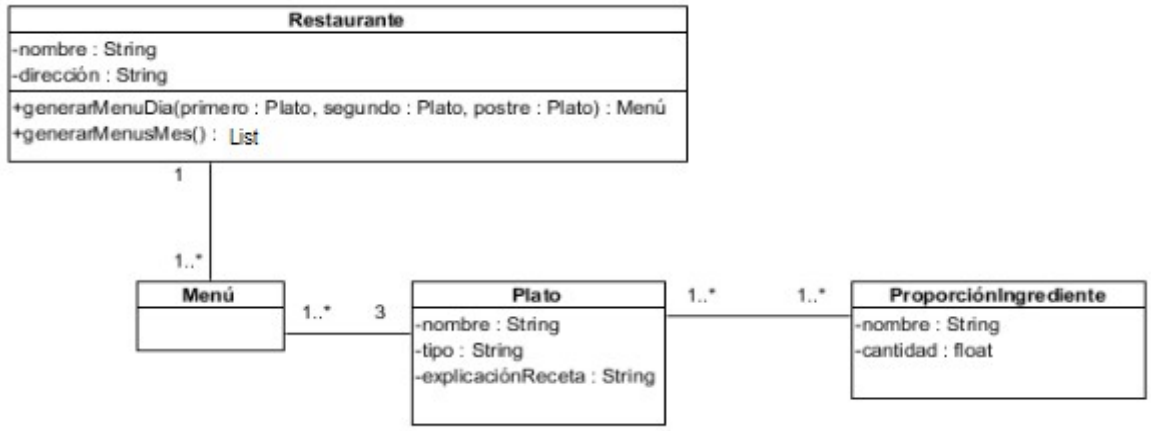
A)



B)



C)



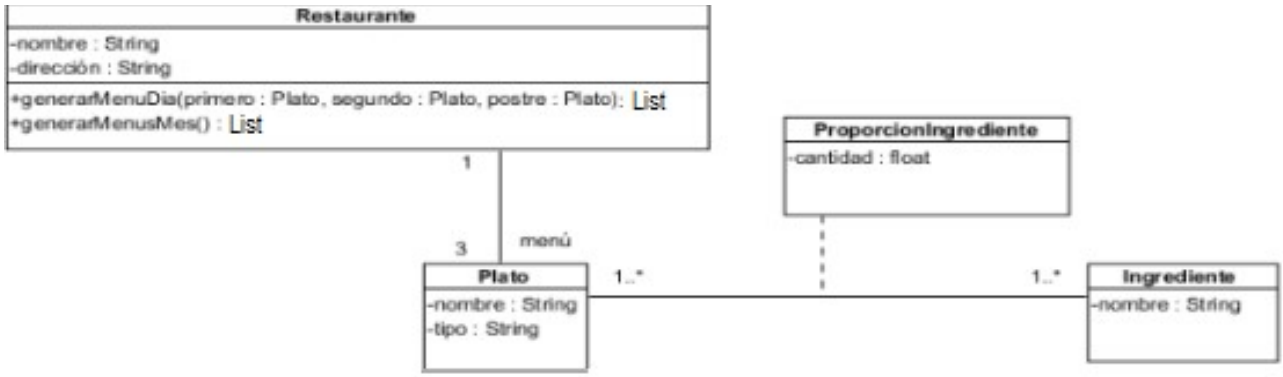
Ejercicio 21. ¿Cómo mejorarías los diseños expuestos en el ejercicio 20 usando de mejor forma la navegabilidad? Indica las navegabilidades que elegirías y explica por qué.

Ejercicio 22. ¿Cómo indicarías en el diagrama de clases que el "tipo" de un Plato puede ser únicamente "primero", "segundo" o "postre"?

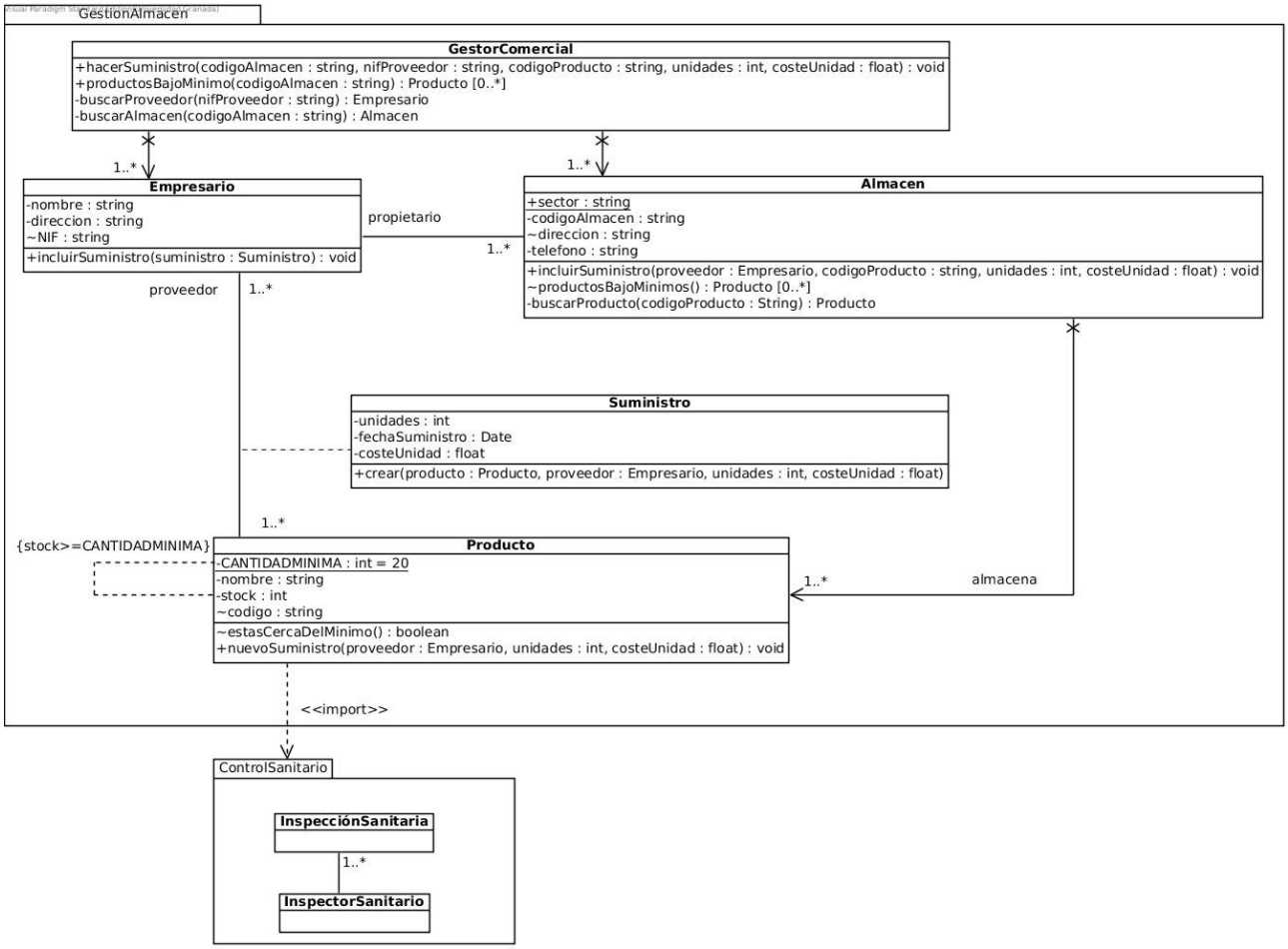
Ejercicio 23. ¿Cómo incluirías la restricción de no repetir ninguna receta en un menú?

Ejercicio 24. Incluye en uno de los diagramas del ejercicio 20 las operaciones y atributos que consideres necesarios para el caso en que estuviésemos interesados en calcular las calorías de los menús.

Ejercicio 25. ¿Es correcto el siguiente diseño según la especificación del ejercicio 14.A? ¿por qué?



Ejercicio 26 (Examen 16/17). Dado el siguiente diagrama de clases:



1) Dibuja cómo representarías en UML la relación entre la clase **Almacen** y una nueva clase **Nave**, sabiendo que un almacén está compuesto por varias naves.

2) Responde verdadero (V) o falso (F).

Si la clase **InspeccionSanitaria** tuviese atributos privados, éstos serían accesibles desde la clase **Producto** puesto que hemos hecho el import del paquete.	
El **stock** de un **Producto** puede ser de menos de 20 unidades.	
Un mismo **Producto** lo pueden suministrar varios **Empresarios**.	
El atributo **sector** de la clase **Almacen** es accesible desde los métodos de la clase **Producto**.	
Del diagrama de clases se deduce que la clase **Suministro** tendrá dos constructores, el indicado en el diagrama y el constructor por defecto.	

3) Teniendo en cuenta sólo la información presentada en el diagrama de clases, implementa en Java la clase **Producto** y en Ruby la clase **Almacen**. Añade e implementa sus constructores para inicializar los atributos. Cuando estos atributos sean colecciones, inicialízalos con la lista vacía. Incluye los métodos consultores (no los modificadores) para todos los atributos.

Ejercicios resueltos

Ejercicio 9

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 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_Vecinos con Ciudadano y con el rol 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 (p.e. presidente o tesorero).

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) private ArrayList <Ciudadano> vecinos;
private Asociacion_Vecinos asociacion;

nombre es un atributo básico, no de referencia.
No hay navegabilidad a Ciudad, luego no incluye atributos de esta clase.

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

M) **Java:**

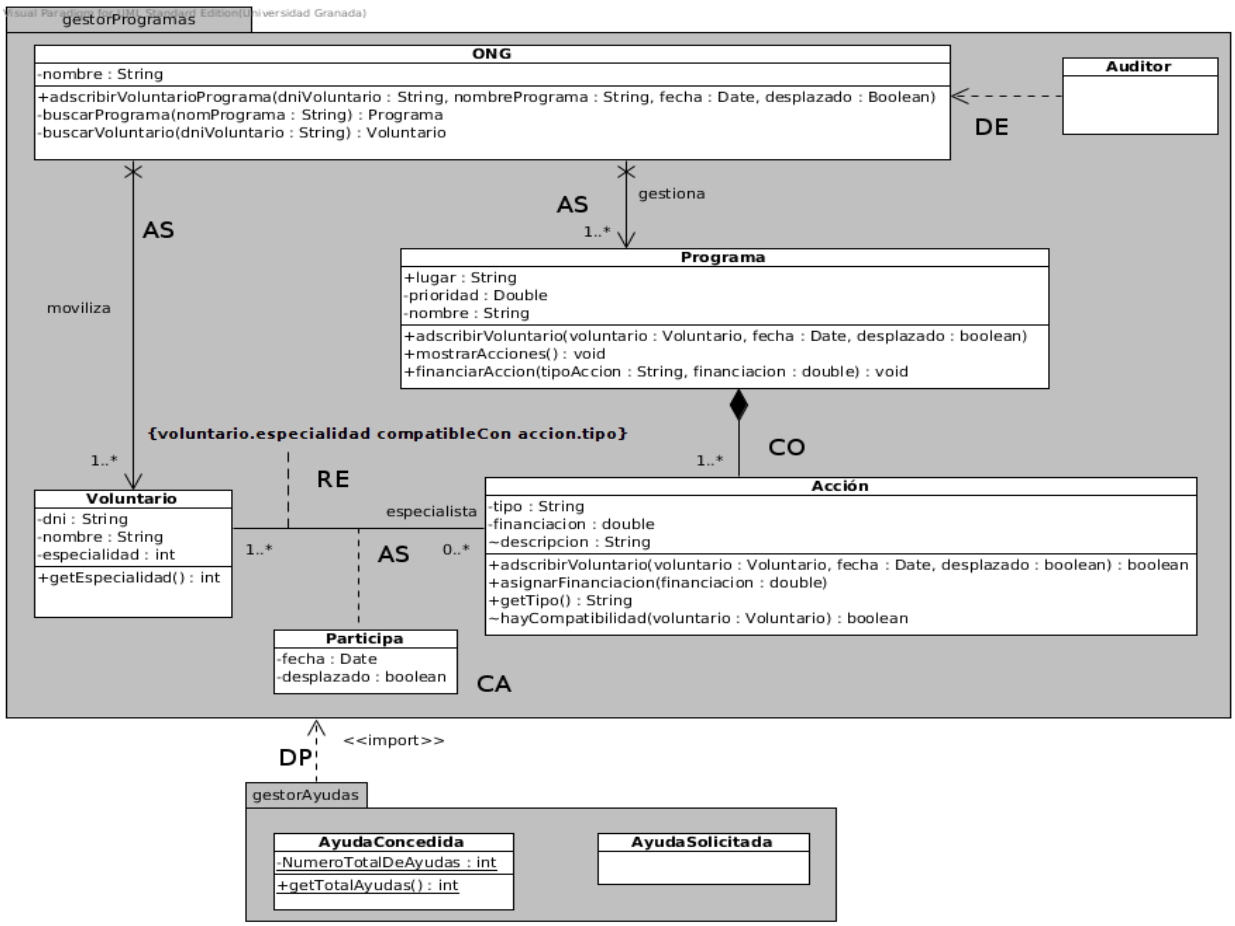
```
package ciudadania;
class Asociacion_Vecinos{
        public String nombre;
        private String cif;
        private Barrio barrio;
        private ArrayList <Ciudadano> socios;
        private ArrayList<Cargo> cargos;
        public Boolean esMiembro(Ciudadano vecino) { }}
package elecciones;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

Ruby:

```
module ciudadania       # en fichero asociacion_vecinos.rb
  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
end

module elecciones  #en fichero elecciones.rb
  require_relative 'ciudadano.rb'
  # suponemos que ciudadano.rb es nombre del fichero que incluye a la clase Ciudadano y que está
  #en la misma carpeta que el fichero actual.
  class Mesa_Electoral
        include ciudadania   #nombre del módulo que se está importando, no se indica una clase
        concreta
        # ciu = ciudadania::Ciudadano.new permitiría crear un objeto
        # puts ciu.nombre  permitiría enviar mensajes al objeto creado
  end
end
```

Ejercicio 11, A)



Ejercicio 11.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
module GestorProgramas
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
end
```

Ruby: AyudaConcedida

```
require_relative 'gestorProgramas.rb'
module gestorAyudas
class AyudaConcedida
    include GestorProgramas
      @@NumeroTotalDeAyudas
      def self.getTotalAyudas
      end
end
end
```

Ejercicio 13

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 objeto reserva está asociado a un único objeto hotel y cliente.