TEMA 1: Introducción a la Orientación a Objetos

Lección 1: Conceptos de OO

1. ¿Qué es un paradigma?
2. ¿Qué es un paradigma de programación?
3. Confecciona una lista con los distintos paradigmas de programación que encuentres describiendo con tus propias palabras sus características principales.
4. ¿Cuál crees que es el elemento esencial de la OO? ¿Por qué?
5. ¿Sería posible un lenguaje OO sin clases?
6. ¿Cuál es la relación entre un método y un envío de mensaje?
7. En clase se ha comentado que la máxima de la POO es que todo es un objeto o un mensaje a un objeto, ¿se cumple dicha máxima en todos los lenguajes de programación? En caso negativo, cita al menos uno y explica por qué.
8. Ordena los conceptos de OO vistos según su relevancia en la OO en general. Razona la respuesta.
9. ¿Qué problemas más importantes puede tener un software con bajo ocultamiento de la información?
10. ¿Qué ventajas más importantes crees que aporta la OO en el diseño de software?
11. ¿Qué diferencia hay entre una estructura de datos y un tipo de dato abstracto (TDA)?
12. ¿Qué diferencia hay entre un TAD y una clase?
13. Principales conceptos introducidos específicamente por la POO, es decir, después de la aparición del TAD.
14. Nombre del concepto al que evoluciona el TAD gracias a la POO.
15. ¿Qué es un objeto?
16. ¿Qué elementos tiene un objeto?
17. ¿Qué es un mensaje?
18. ¿Qué elementos forman parte de un mensaje?
19. ¿Qué pasos son necesarios para resolver un mensaje?
20. ¿Qué es el principio de encapsulación?
21. ¿Qué es una clase?
22. ¿Qué elementos forman parte de una clase?
23. ¿Puede darse la referencias polimórficas a objetos (polimorfismo) sin que se use el mecanismo de ligadura dinámica (tiempo de resolución de referencias en ejecución)? Razona la respuesta.
24. Para que dos objetos de distinta clase puedan ser referenciados por la misma variable ¿Qué deben tener en común? y ¿en qué caso?

Lección 2: Historia de OO

25. Cita algún concepto de programación que haya precedido a la OO y explica en qué medida es un precursor de la OO.
26. ¿En qué año aparece el primer lenguaje de programación orientado a objetos?
27. ¿Qué lenguaje fue?
28. ¿Qué conceptos de OO son introducidos por SIMULA?
29. En qué fila de la tabla de la transparencia 26 (Antecedentes de los lenguajes OO: tipos de datos abstractos) situarías a SIMULA? ¿Qué destacarías de este lenguaje al compararlo con los de la tabla?
30. ¿Qué conceptos, además de los OO, son introducidos por Smalltalk?
31. Establecer el árbol genealógico de C# y de Java, desde los comienzos de la historia de los lenguajes.
32. ¿Qué problema había en los comienzos de uso de métodos de diseño OO?
33. ¿Cuándo se produjo el primer intento de lenguaje de diseño único? y ¿cómo se denominó?
34. ¿Qué características más importantes tiene el método de Booch?
35. Buscar información del método de Yourdon, hacer un pequeño resumen de sus características.
36. ¿Cómo crees que un ingeniero informático como Yourdon pasó de la programación espaguetti al diseño estructurado y luego al diseño OO? ¿Qué ventajas le vio a cada paso?

Lección 3: Métodos y notaciones de OO

37. En las tarjetas CRC que se ¿indica en el apartado de colaboradores?
38. Buscar qué otros formatos para las tarjetas CRC hay.
39. Amplía las tarjetas CRC para incluir otros dos tipos de empleados:
    1. **Empleados por horas** que cobran en función de las horas trabajadas durante el mes correspondiente.
    2. **Empleados a comisión** que cobran un porcentaje del importe total vendido en ese mes.
40. ¿Qué ventajas ha introducido UML como lenguaje de modelado? o mejor dicho ¿Qué intenta proporcionarnos UML para el proceso de desarrollo de un sistema software?
41. Cuando decimos que UML permite construir ¿a qué no estamos refiriendo?
42. ¿Qué se presenta con los diagramas de estructura? y ¿los de comportamiento?
43. Buscar información sobre los diagramas de UML no vistos en clase. haz un resumen de ellos y pon un ejemplo, tratando de explicarlo.

Lección 4: Lenguajes OO

44. Ordena por importancia los criterios de comparación de los lenguajes OO.
45. ¿Qué ventajas presenta los Lenguajes de programación que tratan las clases como objetos sobre los que las tratan como patrones?
46. ¿Qué lenguajes OO crees que implementan perfectamente la máxima "todo es un objeto o un mensaje a un objeto"?
47. ¿Qué es la herencia múltiple? Indica de los siguientes lenguajes, aquéllos que permitan herencia múltiple: Java, c++, Smalltalk, C# y python.
48. ¿Cúal es la diferencia entre polimorfismo y ligadura dinámica?
49. ¿Que significa que el polimorfismo(ligadura dinámica) sea opcional en C++?

Recuerda que debes citar las fuentes que emplees. Si se trata de páginas web, indica el autor, la fecha de consulta y la fecha de última actualización de la página.

Verdes: las que pueden responder con lo que se les explique en clase.

Azules: Las que tienen que investigar un poco y relacionar conceptos.

Rojas: Las que tienen que investigar un poco más.

TEMA 2: Clases y métodos

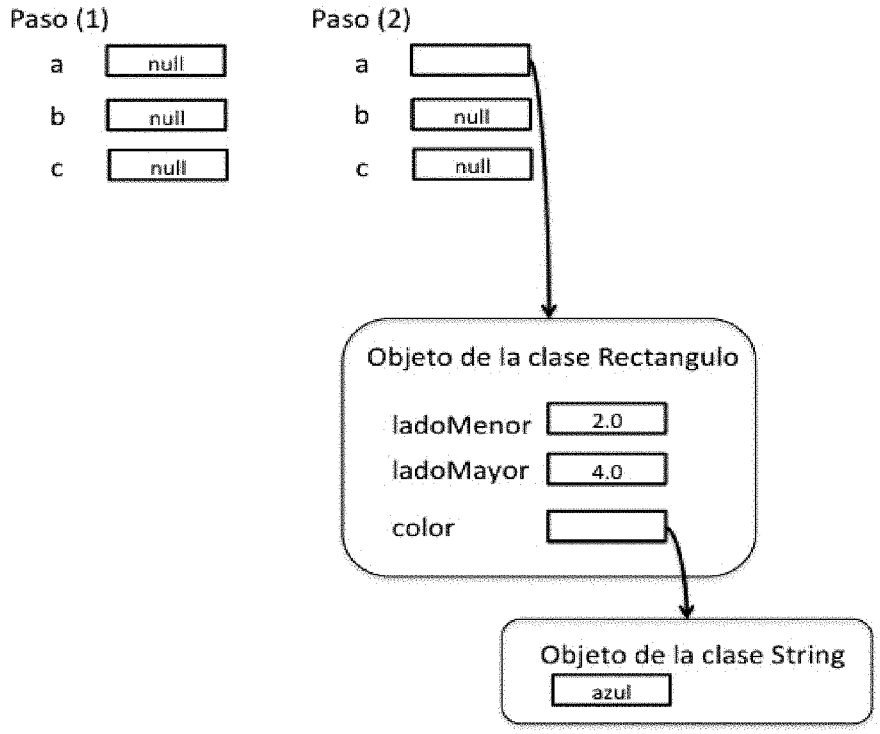
Ejercicio 1. Sea la clase java Rectángulo:

```
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 2. Define en Java y Smalltalk una clase cuyas instancias representen atletas y otra clase cuyas instancias sean equipos de atletas para correr relevos. Incluye los atributos que consideres necesarios. Escribe un programa sencillo que cree un equipo y muestre los atletas que corren en el mismo.

Ejercicio 3. Dada la clase Java:

```
public class Prueba {
     public static int a = 1;
     public int b = 2;
}
```

a. ¿Cuántos atributos tiene? Indica si se trata de atributos de instancia o de clase.

b. Indica cuál es el estado de obj1 y obj2 después de ejecutar la siguiente secuencia de código:

```
Prueba obj1 = new Prueba();
Prueba obj2 = new Prueba();
obj1.a = 3;
obj1.b = 4;
obj2.a = 5;
obj2.b = 6;
```

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

Ejercicio 4. ¿Quién debe tener la responsabilidad de responder a los mensajes que se corresponden con los llamados métodos de clase? ¿Se hace así en Smalltalk? ¿Y en Java?

Ejercicio 5. Razona si las siguientes afirmaciones son ciertas o falsas:

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) / "self" (Smalltalk) puede emplearse tanto en métodos de clase como de instancia.

Ejercicio 6. Razona si las afirmaciones siguientes sobre la clase Empleado son verdaderas o falsas:

```
Object subclass: #Empleado
instanceVariableinstanceVariableNames: 'dni nombre'
classVariableNames:'PorcentajeRetencion'
poolDictionaries:'MiPoolDiccionario'
Smalltalk at:#MiPoolDiccionario put:Dictionary new.
MiPoolDiccionario at:#Var1 put:valor
```

a. Empleado es una variable global.
b. Smalltalk es una variable global.
c. Smalltalk inspect permite ver todas las variables globales.
d. PorcentajeRetencion es una variable de clase.
e. dni es una variable de instancia.
f. dni es una variable privada.
g. MiPoolDiccionario es una variable global.
h. Var1 es una variable semi-global.

Ejercicio 7. ¿Cuántos constructores distintos puede tener una clase? ¿qué los diferencia?

Ejercicio 8. Indica el tipo de método Smalltalk que se corresponde con las siguientes expresiones:

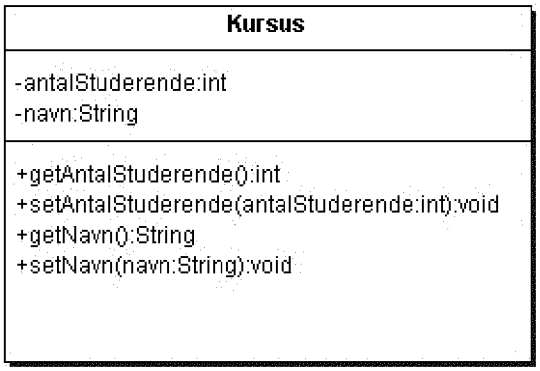
```
45 factorial.
'hola' outputToPrinter.
3+4.
var at:3.
var at:3 put:$x.
```

Ejercicio 9. En el código *claseAlumno1.txt* (está en SWAD, corresponde a la sesión 1 de la práctica 1) ¿con qué especificador de acceso se han declarado los atributos y métodos? ¿crees que esta elección es la óptima? En caso negativo, indica qué especificadores de acceso emplearías.

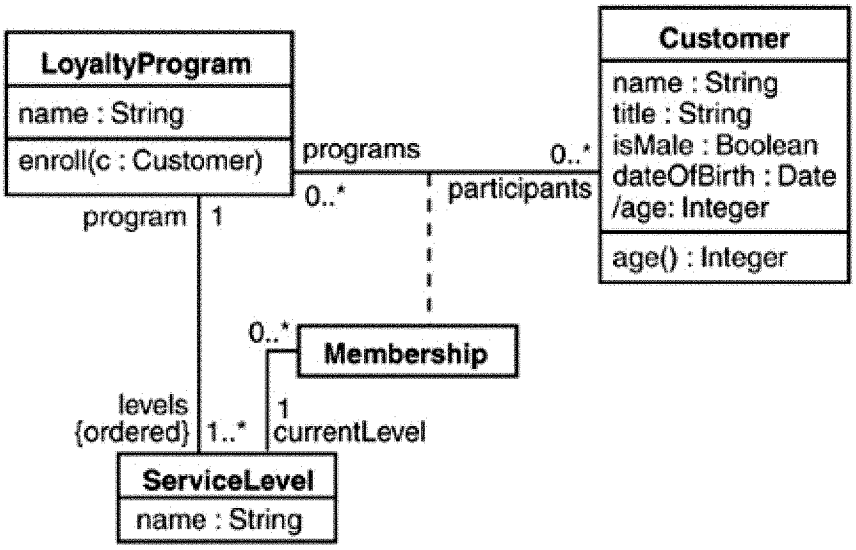
Ejercicio 10. ¿Qué mecanismos tienen Java y Smalltalk para ocultación de información?

Ejercicio 11. En "ejerciciosPruebasAcceso" hay un proyecto NetBeans con dos paquetes de clases. Cada una de estas clases posee un método main, ejecútalo y explica qué sucede. En el caso de que se den errores, explica a qué se deben y cómo los solucionarías variando las restricciones de acceso.

Ejercicio 12. UML es un lenguaje "universal". Escribe el código Java y Smalltalk correspondiente a la declaración de la siguiente clase en danés (con la declaración de atributos así como la cabecera de sus métodos).

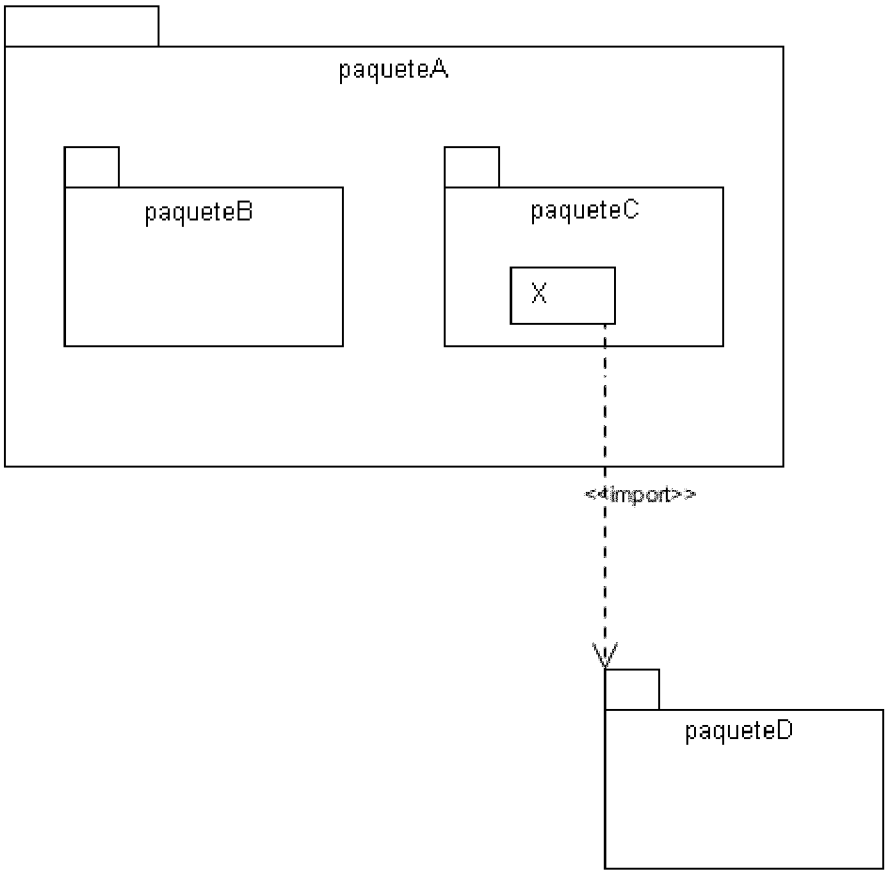


Ejercicio 13: Escribe el código Java y Smalltalk correspondiente a las clases que aparecen en el siguiente diagrama de clases sin olvidar los atributos de referencia y haciendo uso del nombre de los roles que figuran en el diagrama.



Ejercicio 14: En el código que has generado, ¿qué implicación tendrá la restricción {ordered}? ¿qué harías para asegurate de que se cumple?

Ejercicio 15: Dado el siguiente diagrama de paquetes:

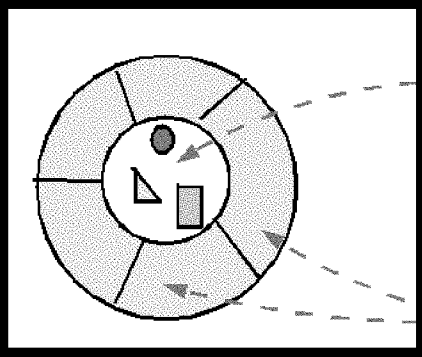


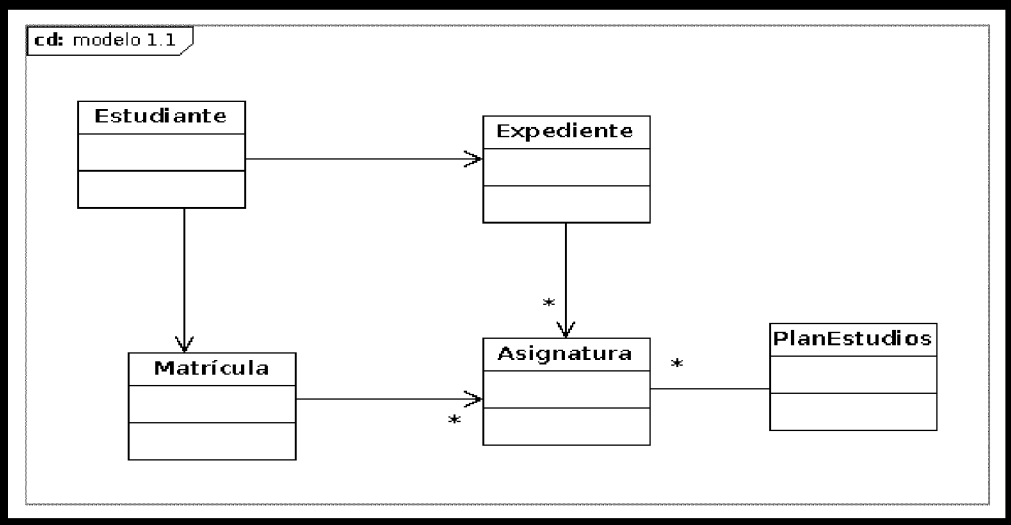
a) Indica cómo construirías esta estructura en Java y Smalltalk.

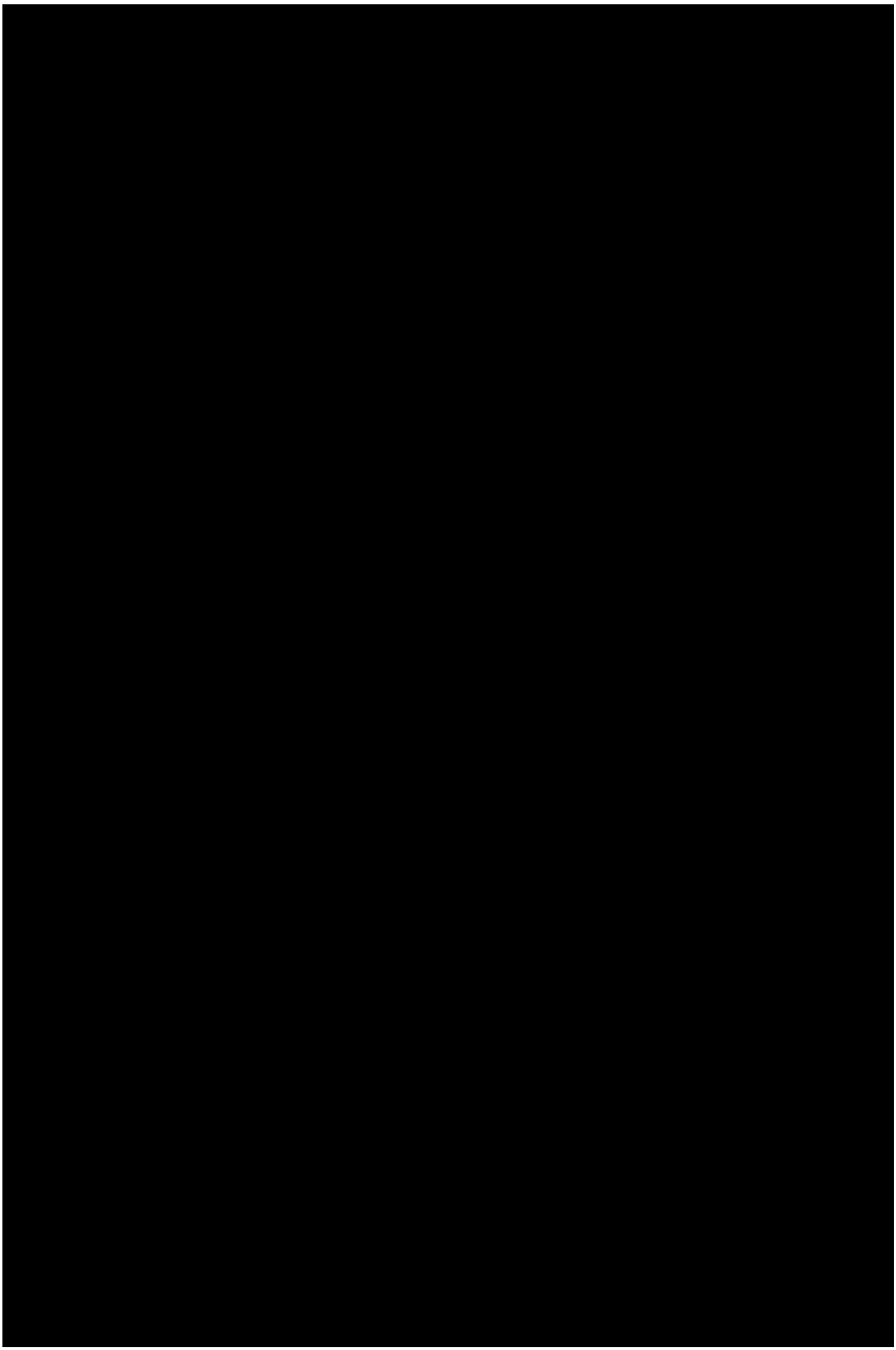
b) Justifica si sería accesible un atributo de la clase X declarado con visibilidad de paquete desde:

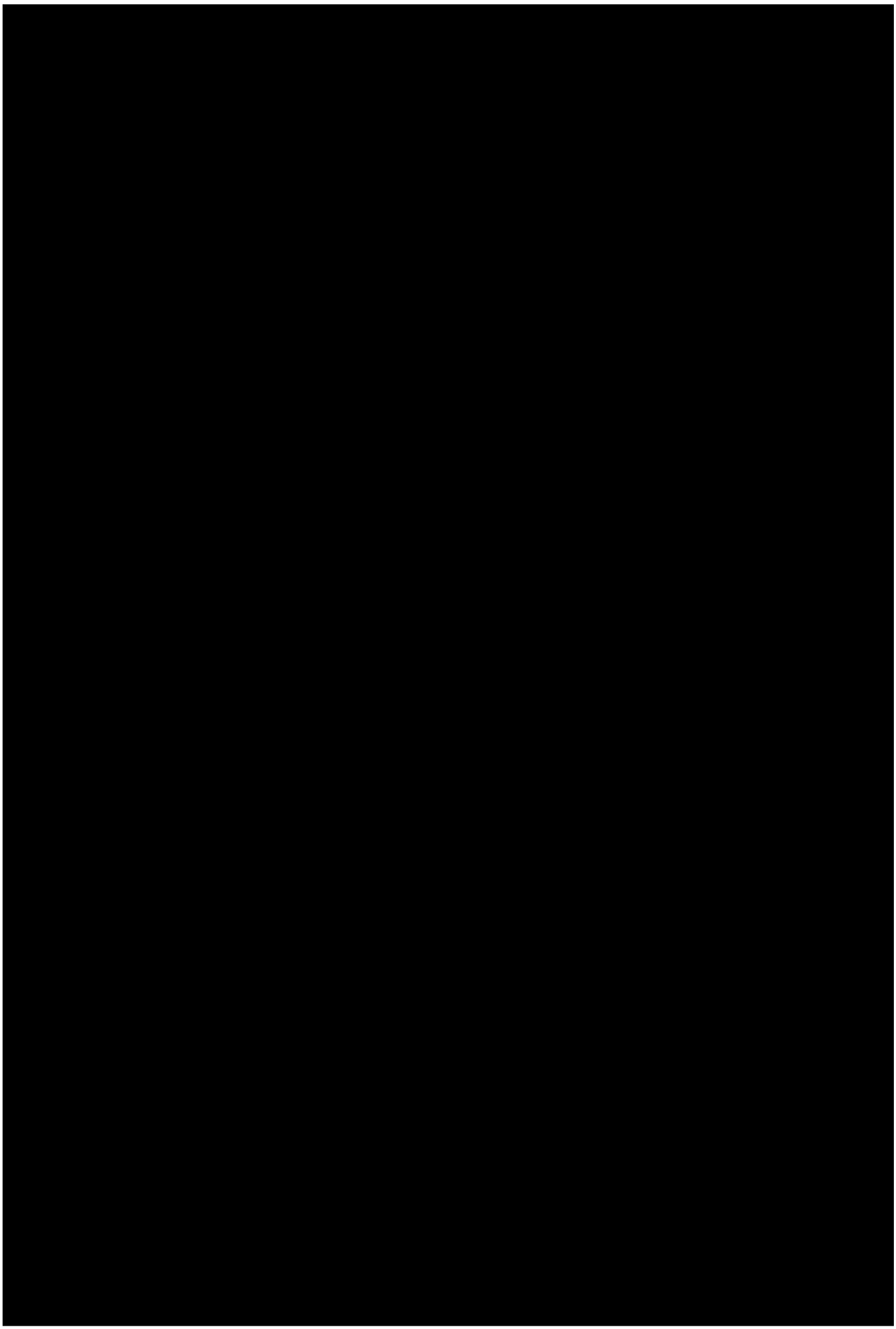
- Una clase delpaqueteB
- Una clase de paqueteA que no esté en paqueteB ni en paqueteC
- Una clase de paqueteD

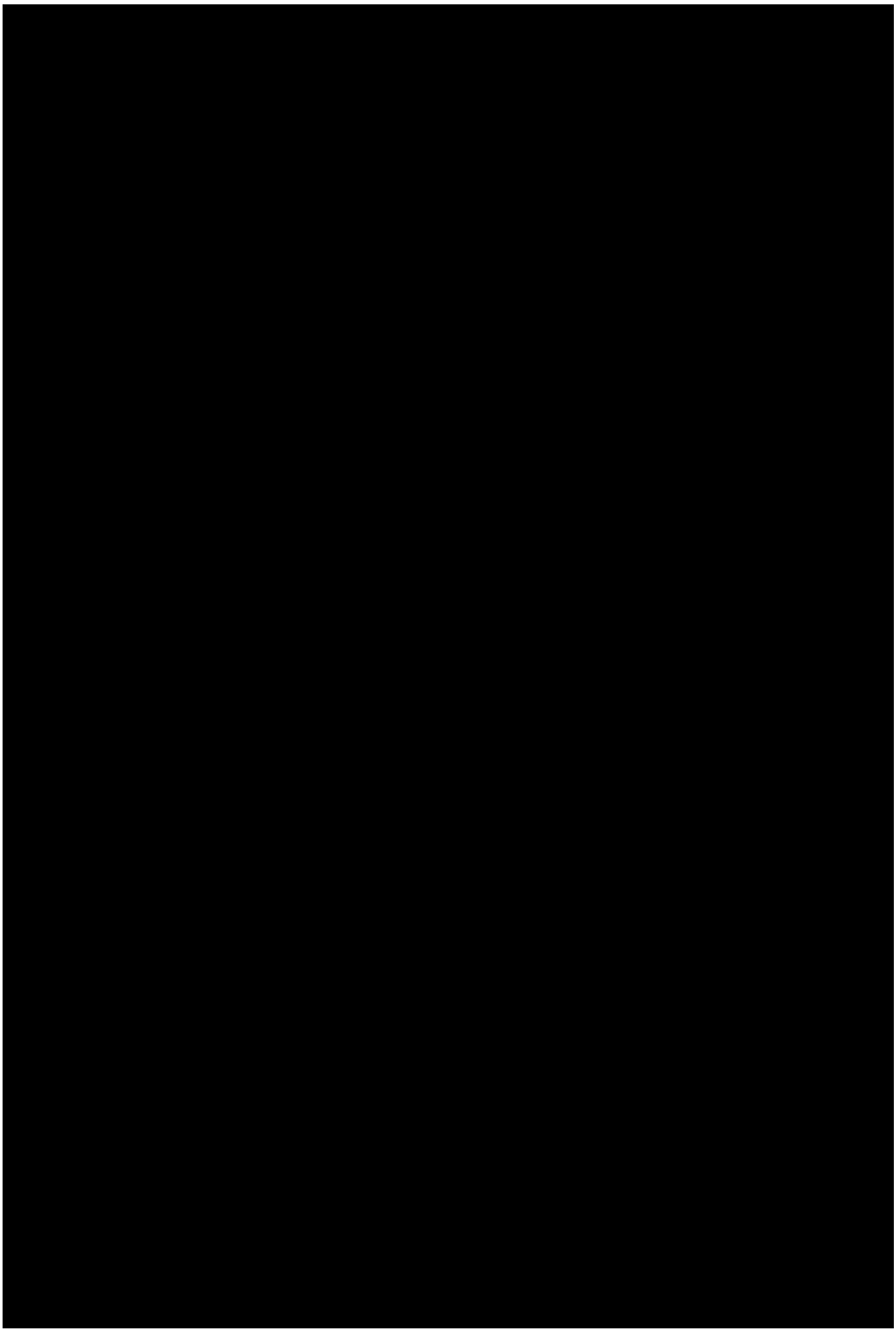
c) ¿Y si se declarase "protected"?

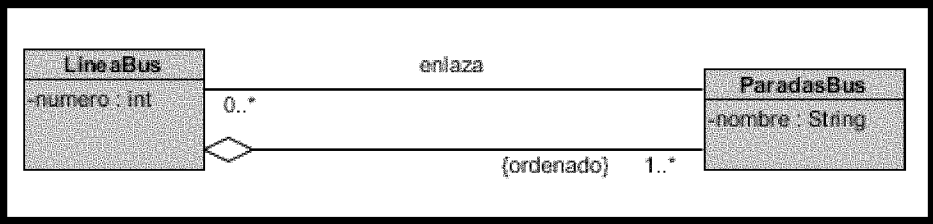



cd: modelo 1.1
Estudiante
Expediente
Matrícula
Asignatura
PlanEstudios
*
*
*
*

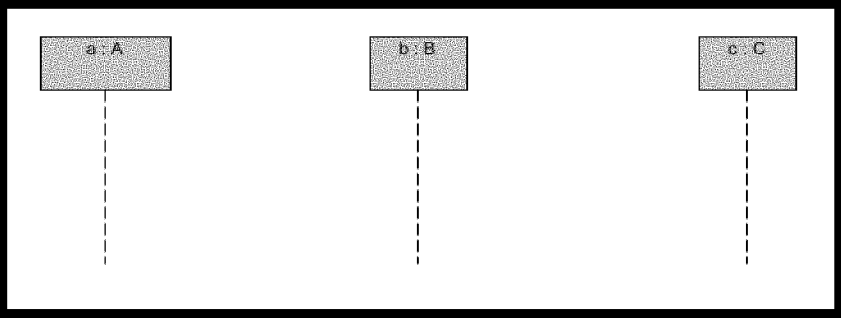


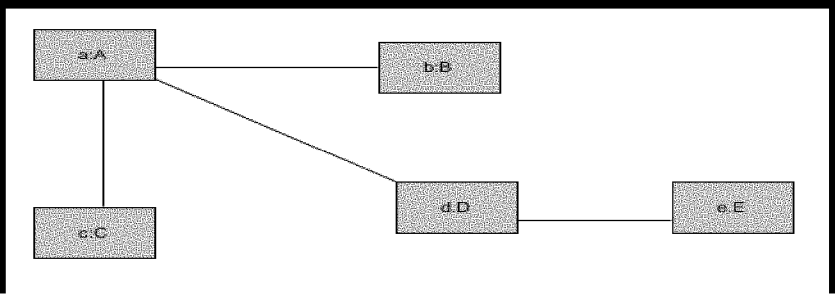


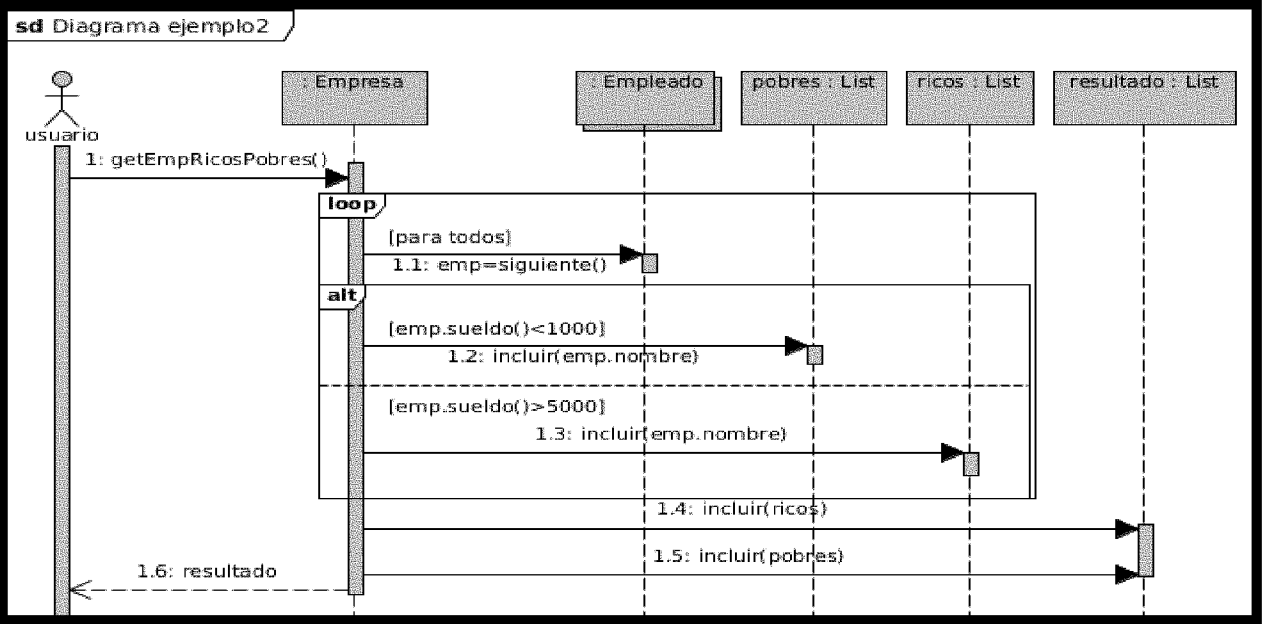


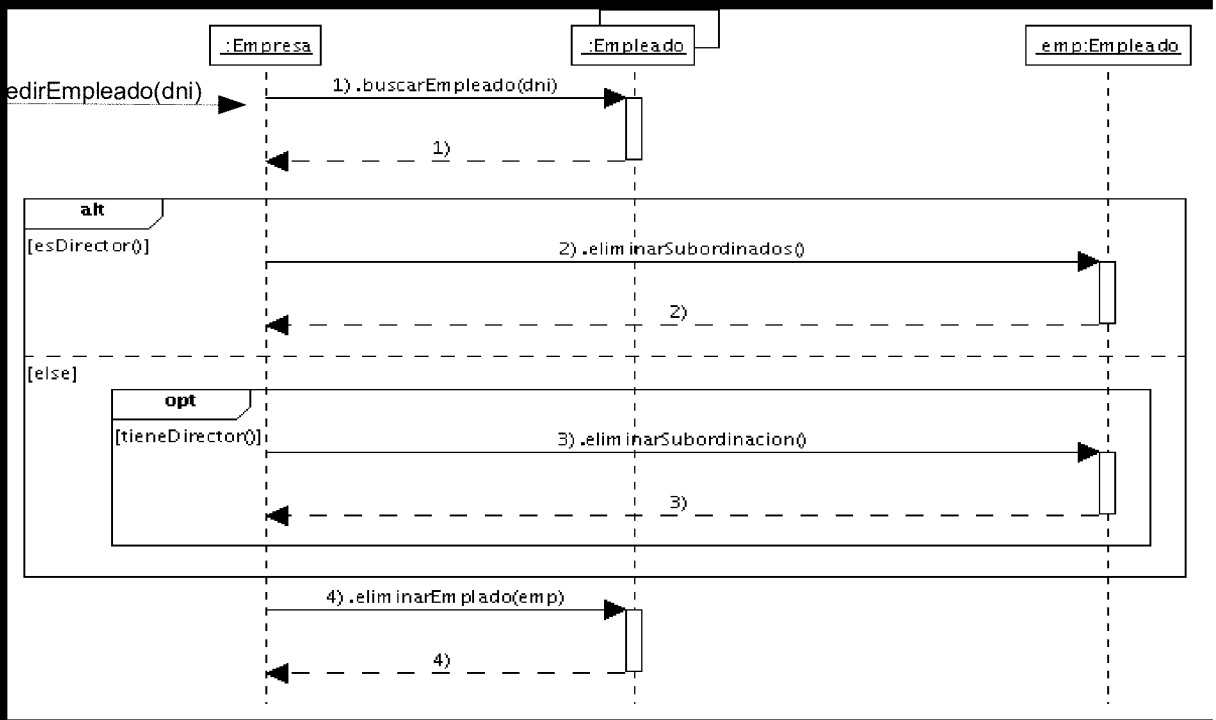

LineaBus
-numero : int
enlaza
0..*
{ordenado}
1..*
ParadasBus
-nombre : String

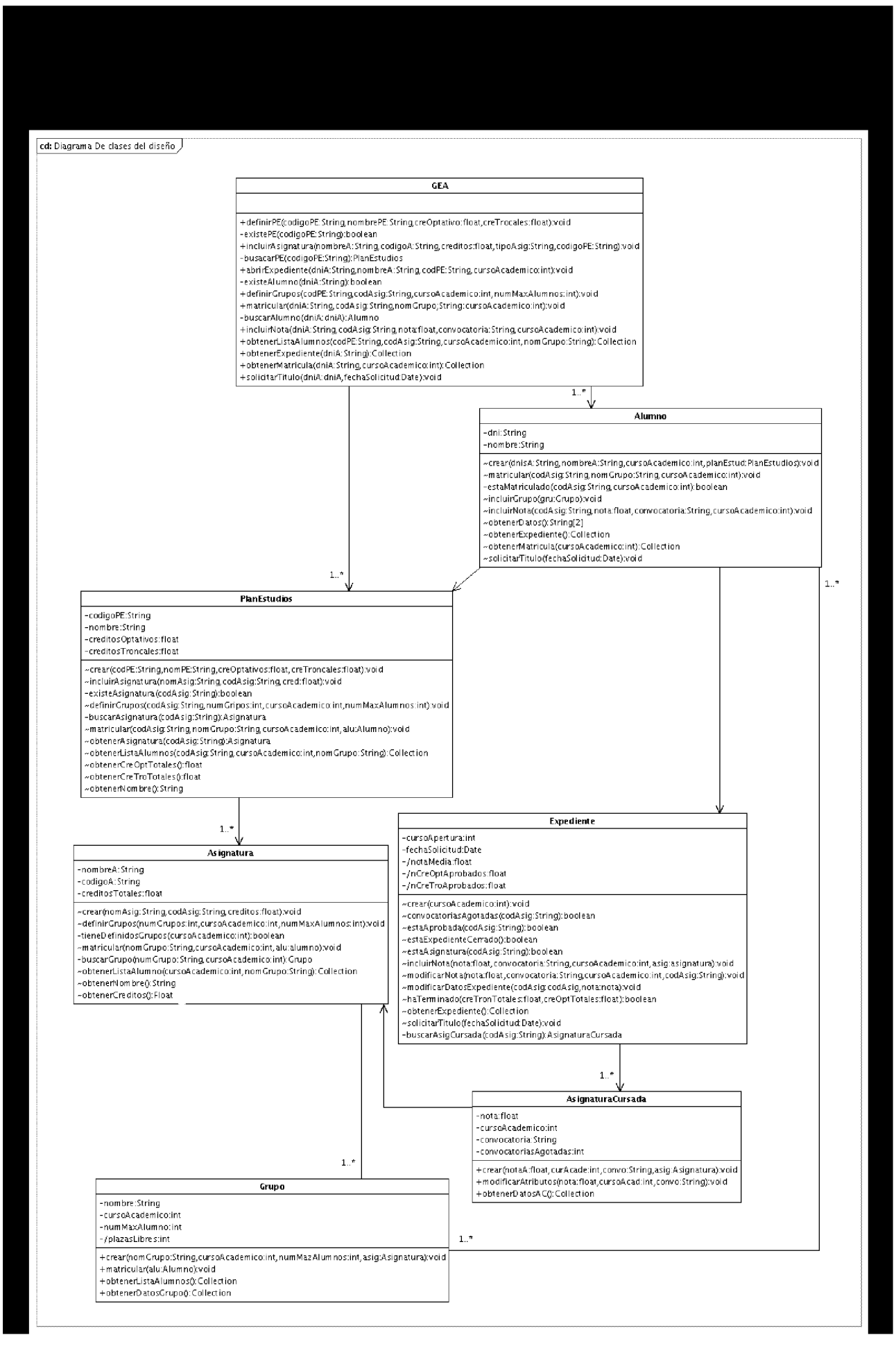

```
4 + 7 * 5 + 2
10 raizcuadrada 2 + 4 agrupa
edades at: 'Juan' put: (edades at:'Pedro')
edades at:'Juan' put: edades at:'Pedro'
```

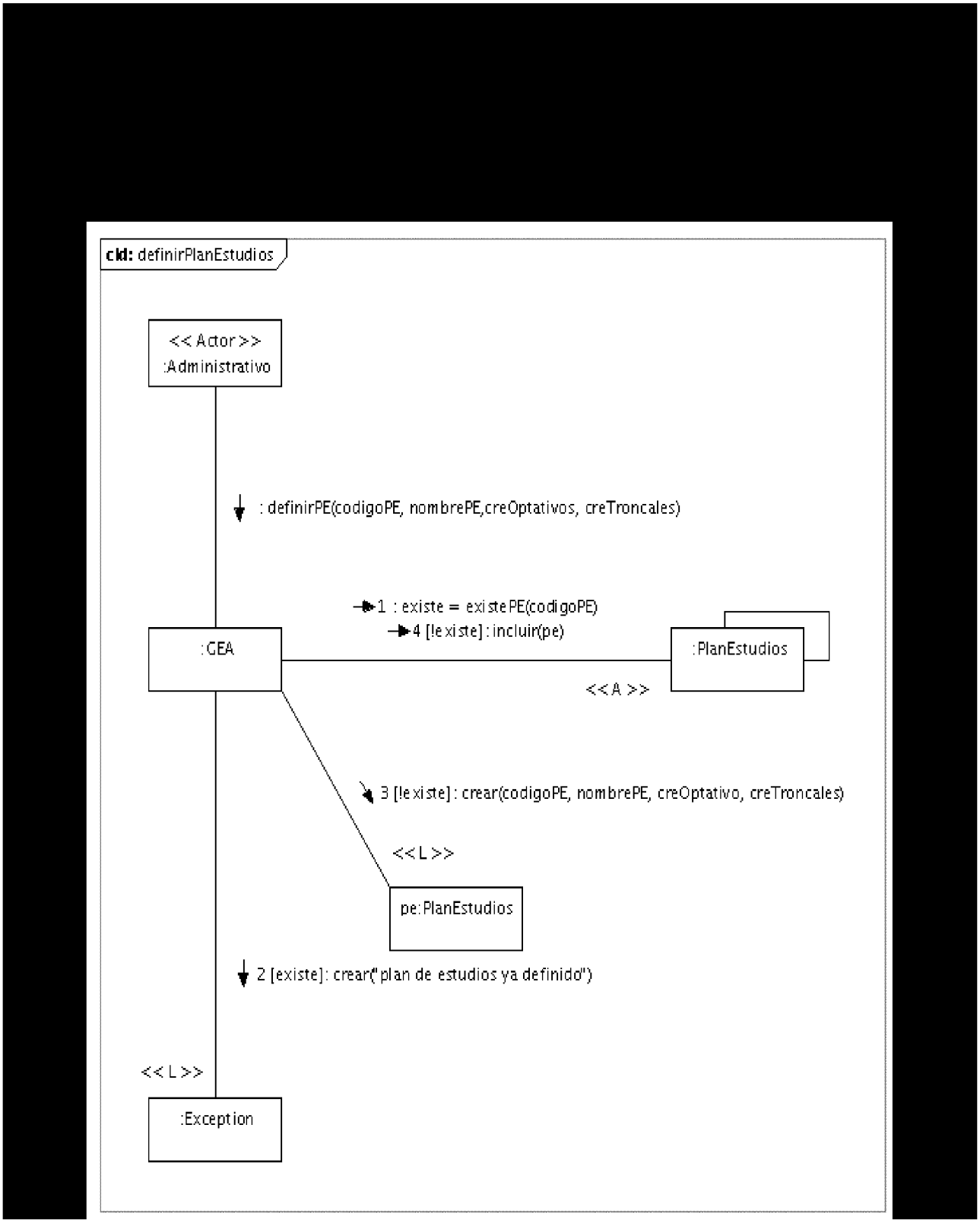

a : A
b : B
c : C

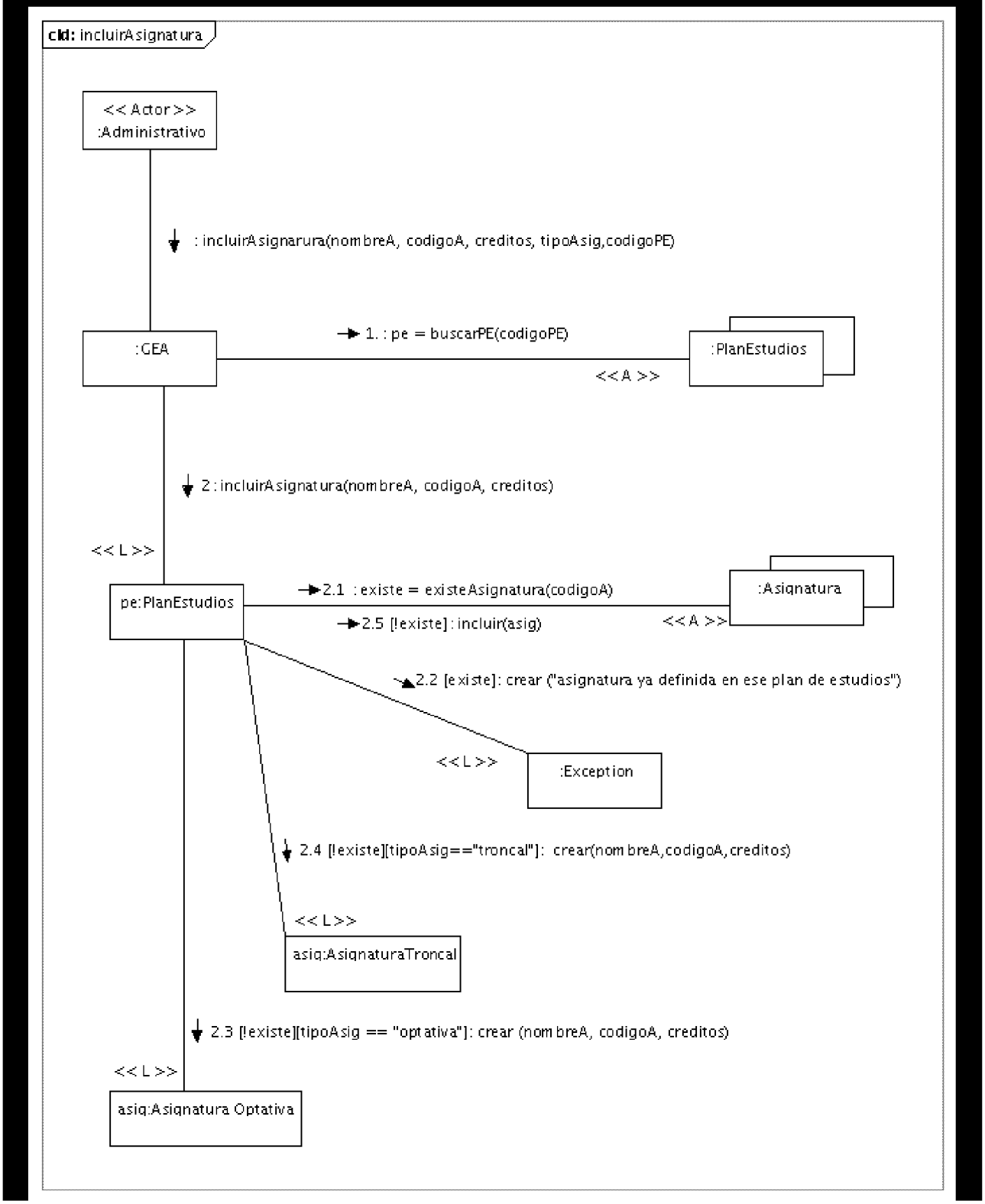

a:A
b:B
c:C
d:D
e:E

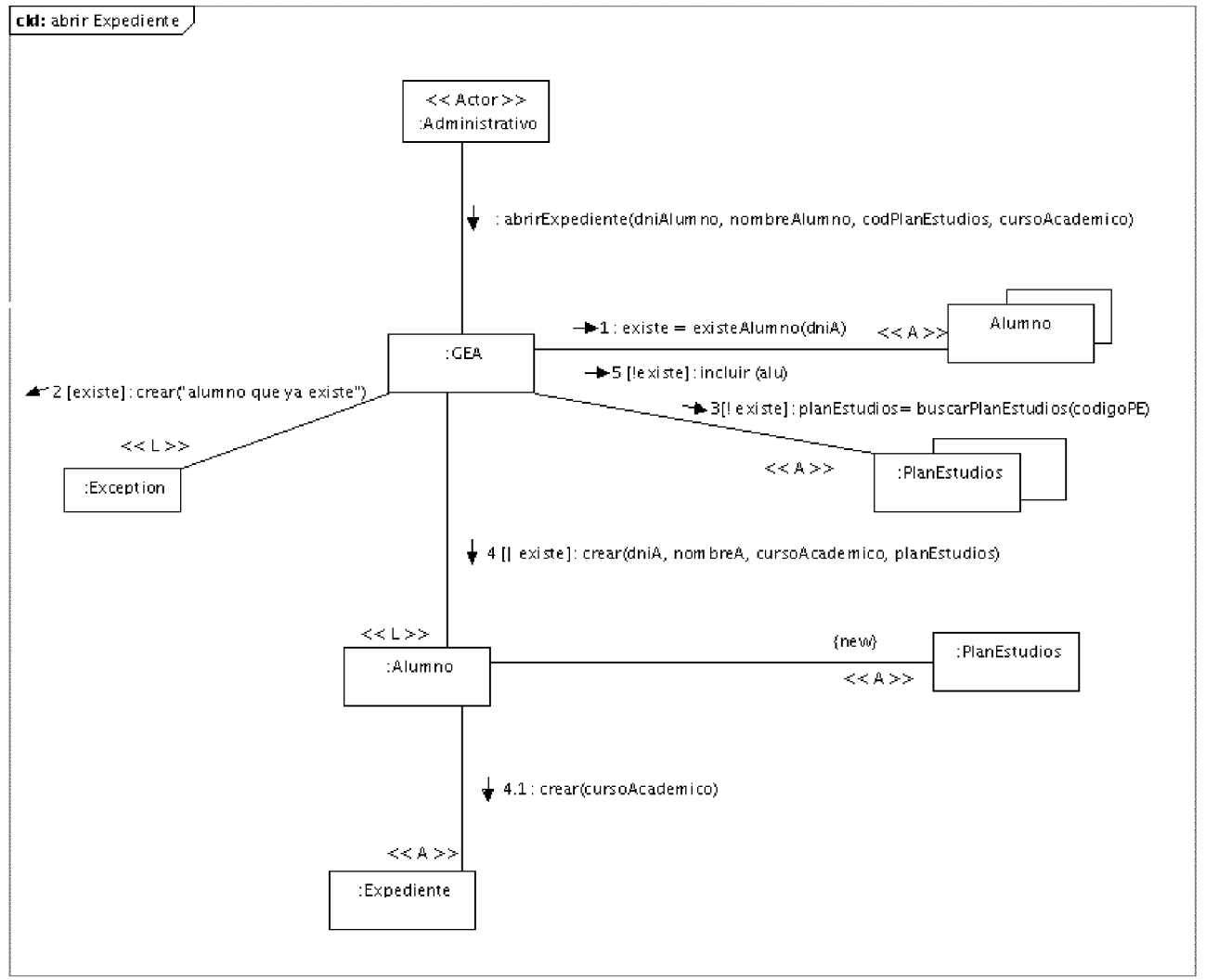

sd Diagrama ejemplo2
usuario
: Empresa
: Empleado
pobres : List
ricos : List
resultado : List
1: getEmpRicosPobres()
loop
[para todos]
1.1: emp=siguiente()
alt
[emp.sueldo()<1000]
1.2: incluir(emp.nombre)
[emp.sueldo()>5000]
1.3: incluir(emp.nombre)
1.4: incluir(ricos)
1.5: incluir(pobres)
1.6: resultado

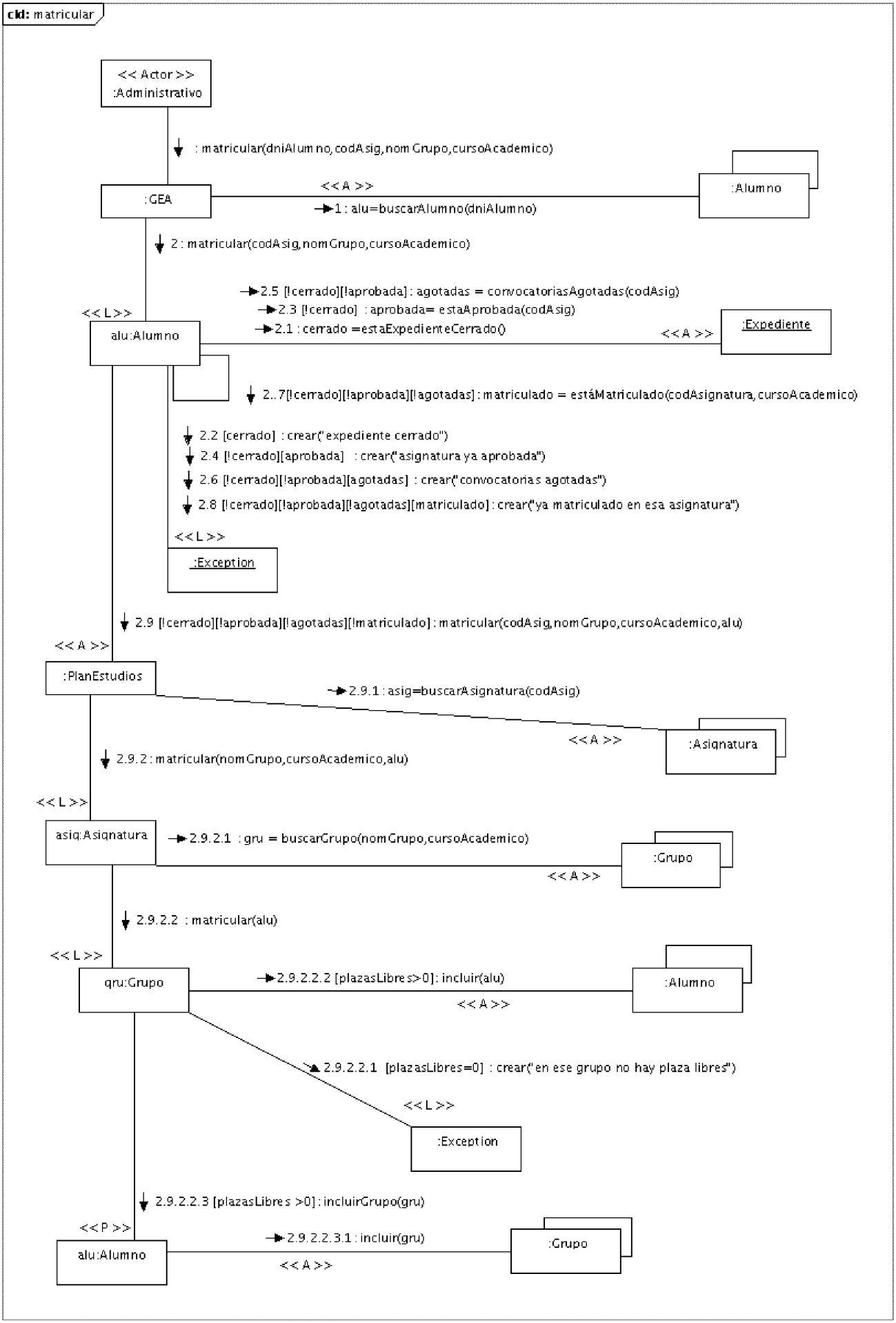

:Empresa
:Empleado
emp:Empleado
edirEmpleado(dni)
1).buscarEmpleado(dni)
1)
alt
[esDirector()]
2).eliminarSubordinados()
2)
[else]
opt
[tieneDirector()]
3).eliminarSubordinacion()
3)
4).eliminarEmplado(emp)
4)

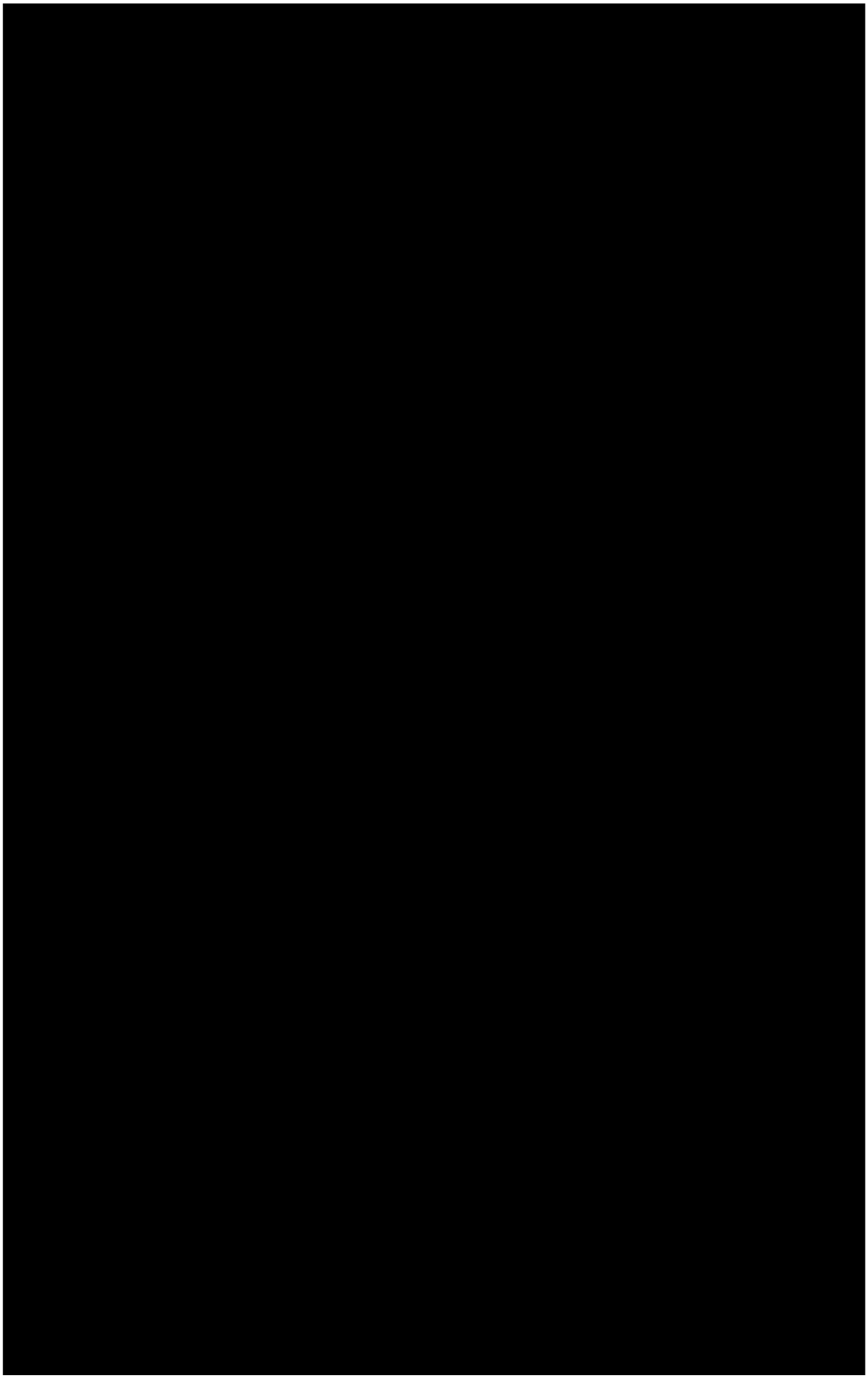

cd: Diagrama De clases del diseño
GEA
+definirPE(codigoPE:String,nombrePE:String,creOptativo:float,creTrocales:float):void
-existePE(codigoPE:String):boolean
+incluirAsignatura(nombreA:String,codigoA:String,creditos:float,tipoAsig:String,codigoPE:String):void
-busacarPE(codigoPE:String):PlanEstudios
+abrirExpediente(dniA:String,nombreA:String,codPE:String,cursoAcademico:int):void
-existeAlumno(dniA:String):boolean
+definirGrupos(codPE:String,codAsig:String,cursoAcademico:int,numMaxAlumnos:int):void
+matricular(dniA:String,codAsig:String,nomGrupo;String:cursoAcademico:int):void
-buscarAlumno(dniA:dniA):Alumno
+incluirNota(dniA:String,codAsig:String,nota:float,convocatoria:String,cursoAcademico:int):void
+obtenerListaAlumnos(codPE:String,codAsig:String,cursoAcademico:int,nomGrupo:String):Collection
+obtenerExpediente(dniA:String):Collection
+obtenerMatricula(dniA:String,cursoAcademico:int):Collection
+solicitarTitulo(dniA:dniA,fechaSolicitud:Date):void
1..*
Alumno
-dni:String
-nombre:String
~crear(dnisA:String,nombreA:String,cursoAcademico:int,planEstud:PlanEstudios):void
~matricular(codAsig:String,nomGrupo:String,cursoAcademico:int):void
-estaMatriculado(codAsig:String,cursoAcademico:int):boolean
~incluirGrupo(gru:Grupo):void
~incluirNota(codAsig:String,nota:float,convocatoria:String,cursoAcademico:int):void
~obtenerDatos():String[2]
~obtenerExpediente():Collection
~obtenerMatricula(cursoAcademico:int):Collection
~solicitarTitulo(fechaSolicitud:Date):void
1..*
1..*
PlanEstudios
-codigoPE:String
-nombre:String
-creditosOptativos:float
-creditosTroncales:float
~crear(codPE:String,nomPE:String,creOptativos:float,creTroncales:float):void
~incluirAsignatura(nomAsig:String,codAsig:String,cred:float):void
-existeAsignatura(codAsig:String):boolean
~definirGrupos(codAsig:String,numGrupos:int,cursoAcademico:int,numMaxAlumnos:int):void
-buscarAsignatura(codAsig:String):Asignatura
~matricular(codAsig:String,nomGrupo:String,cursoAcademico:int,alu:Alumno):void
~obtenerAsignatura(codAsig:String):Asignatura
~obtenerListaAlumnos(codAsig:String,cursoAcademico:int,nomGrupo:String):Collection
~obtenerCreOptTotales():float
~obtenerCreTroTotales():float
~obtenerNombre():String
1..*
Expediente
-cursoApertura:int
-fechaSolicitud:Date
-/notaMedia:float
-/nCreOptAprobados:float
-/nCreTroAprobados:float
~crear(cursoAcademico:int):void
~convocatoriasAgotadas(codAsig:String):boolean
~estaAprobada(codAsig:String):boolean
~estaExpedienteCerrado():boolean
~estaAsignatura(codAsig:String):boolean
~incluirNota(nota:float,convocatoria:String,cursoAcademico:int,asig:asignatura):void
~modificarNota(nota:float,convocatoria:String,cursoAcademico:int,codAsig:String):void
~modificarDatosExpediente(codAsig:codAsig,nota:nota):void
~haTerminado(creTronTotales:float,creOptTotales:float):boolean
~obtenerExpediente():Collection
~solicitarTitulo(fechaSolicitud:Date):void
-buscarAsigCursada(codAsig:String):AsignaturaCursada
Asignatura
-nombreA:String
-codigoA:String
-creditosTotales:float
~crear(nomAsig:String,codAsig:String,creditos:float):void
~definirGrupos(numGrupos:int,cursoAcademico:int,numMaxAlumnos:int):void
-tieneDefinidosGrupos(cursoAcademico:int):boolean
~matricular(nomGrupo:String,cursoAcademico:int,alu:alumno):void
-buscarGrupo(numGrupo:String,cursoAcademico:int):Grupo
~obtenerListaAlumno(cursoAcademico:int,nomGrupo:String):Collection
~obtenerNombre():String
~obtenerCreditos():Float
1..*
AsignaturaCursada
-nota:float
-cursoAcademico:int
-convocatoria:String
-convocatoriasAgotadas:int
+crear(notaA:float,curAcade:int,convo:String,asig:Asignatura):void
+modificarAtributos(nota:float,cursoAcad:int,convo:String):void
+obtenerDatosACQ:Collection
1..*
Grupo
-nombre:String
-cursoAcademico:int
-numMaxAlumno:int
-/plazasLibres:int
+crear(nomGrupo:String,cursoAcademico:int,numMazAlumnos:int,asig:Asignatura):void
+matricular(alu:Alumno):void
+obtenerListaAlumnos():Collection
+obtenerDatosGrupo():Collection
1..*

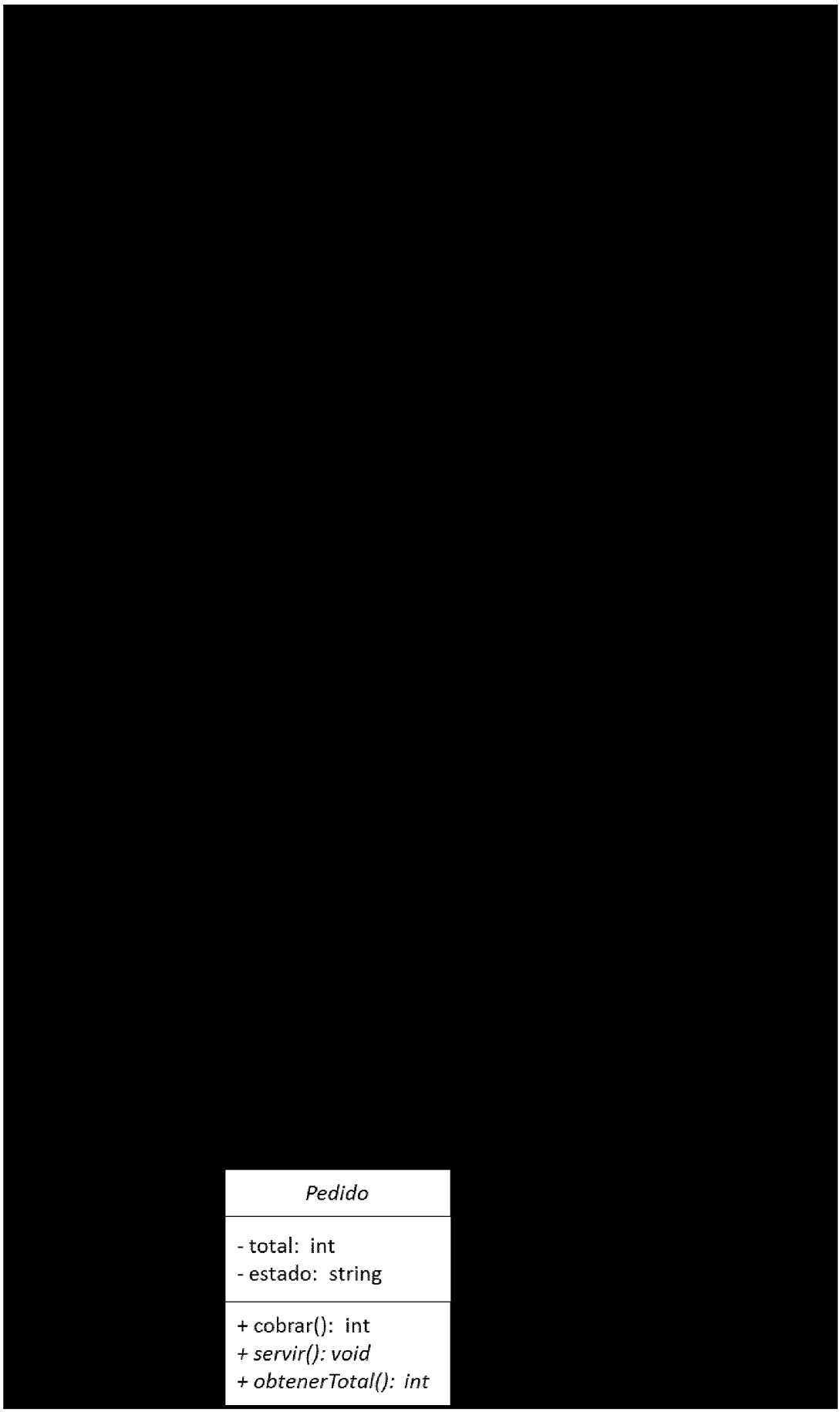

cld: definirPlanEstudios
<< Actor >>
:Administrativo
: definirPE(codigoPE, nombrePE,creOptativos, creTroncales)
1 : existe = existePE(codigoPE)
4 [!existe] : incluir(pe)
:GEA
:PlanEstudios
<< A >>
3 [!existe] : crear(codigoPE, nombrePE, creOptativo, creTroncales)
<< L >>
pe:PlanEstudios
2 [existe]: crear("plan de estudios ya definido")
<< L >>
:Exception

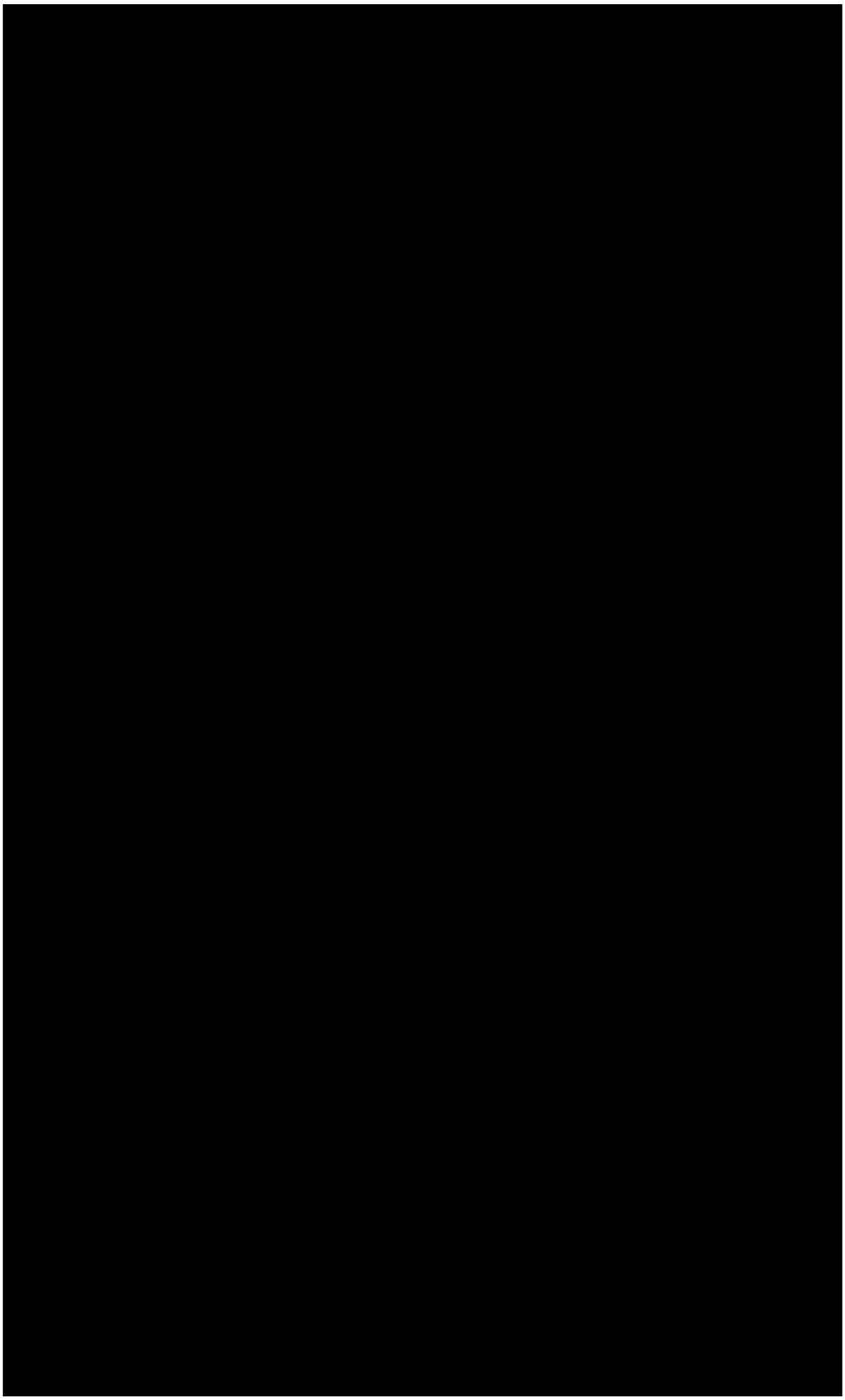

cld: incluirAsignatura
<< Actor >>
:Administrativo
: incluirAsignarura(nombreA, codigoA, creditos, tipoAsig,codigoPE)
:GEA
1. : pe = buscarPE(codigoPE)
<< A >>
:PlanEstudios
2 : incluirAsignatura(nombreA, codigoA, creditos)
<< L >>
pe:PlanEstudios
2.1 : existe = existeAsignatura(codigoA)
2.5 [!existe] : incluir(asig)
<< A >>
:Asignatura
2.2 [existe]: crear ("asignatura ya definida en ese plan de estudios")
<< L >>
:Exception
2.4 [!existe][tipoAsig=="troncal"]: crear(nombreA,codigoA,creditos)
<< L >>
asig:AsignaturaTroncal
2.3 [!existe][tipoAsig == "optativa"]: crear (nombreA, codigoA, creditos)
<< L >>
asig:Asignatura Optativa

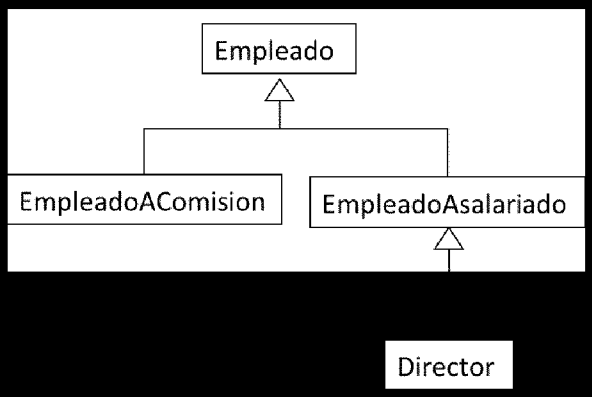

cld: abrir Expediente
<< Actor >>
:Administrativo
: abrirExpediente(dniAlumno, nombreAlumno, codPlanEstudios, cursoAcademico)
:GEA
1 : existe = existeAlumno(dniA)
<< A >>
Alumno
5 [!existe] : incluir (alu)
2 [existe] : crear("alumno que ya existe")
3[! existe] : planEstudios= buscarPlanEstudios(codigoPE)
<< L >>
:Exception
<< A >>
:PlanEstudios
4 [! existe] : crear(dniA, nombreA, cursoAcademico, planEstudios)
<< L >>
:Alumno
(new)
:PlanEstudios
<< A >>
4.1 : crear(cursoAcademico)
<< A >>
:Expediente

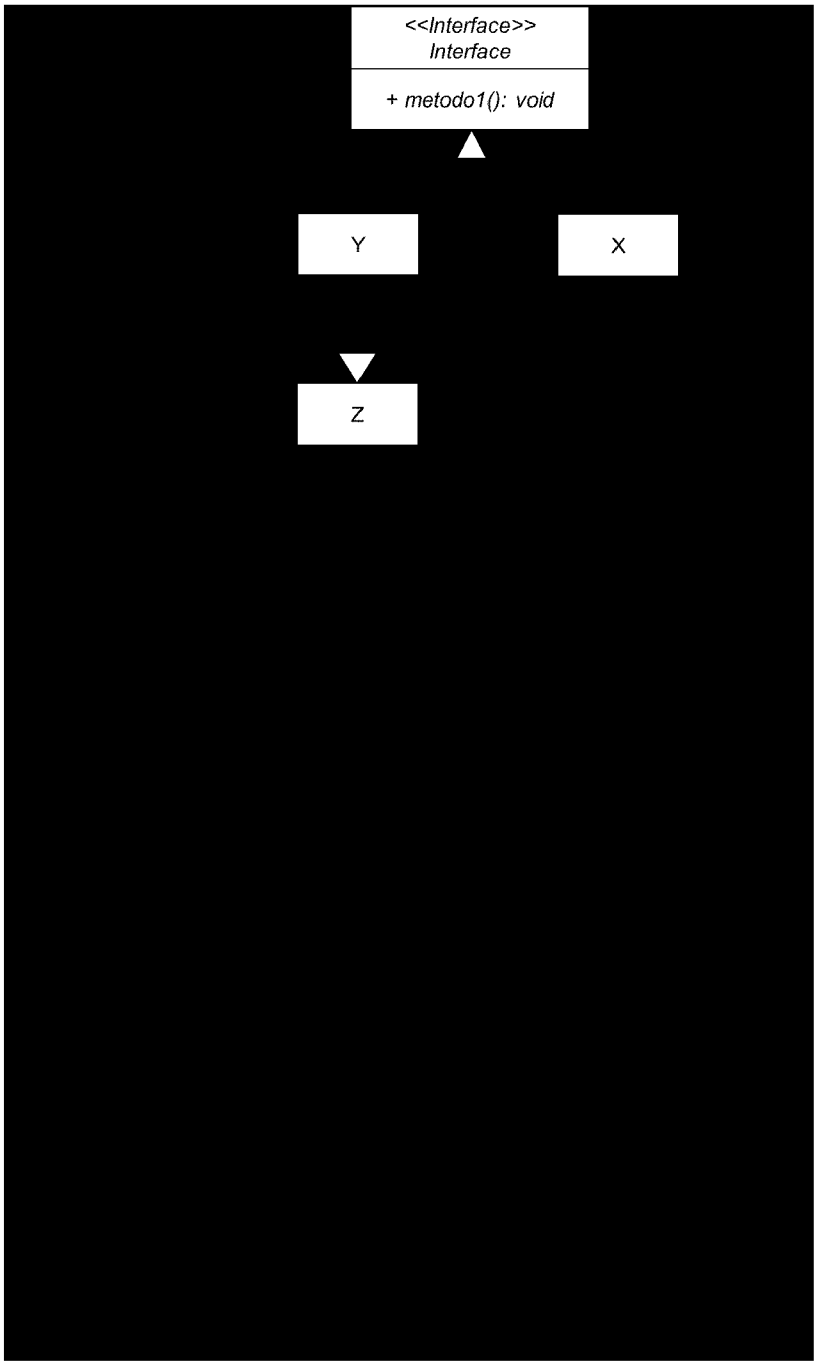

cld: matricular
<< Actor >>
:Administrativo
: matricular(dniAlumno,codAsig,nomGrupo,cursoAcademico)
:GEA
<< A >>
1 : alu=buscarAlumno(dniAlumno)
:Alumno
2 : matricular(codAsig,nomGrupo,cursoAcademico)
2.5 [!cerrado][!aprobada] : agotadas = convocatoriasAgotadas(codAsig)
2.3 [!cerrado] : aprobada= estaAprobada(codAsig)
2.1 : cerrado =estaExpedienteCerrado()
<< L >>
alu:Alumno
<< A >>
:Expediente
2..7[!cerrado][!aprobada][!agotadas] : matriculado = estáMatriculado(codAsignatura,cursoAcademico)
2.2 [cerrado] : crear("expediente cerrado")
2.4 [!cerrado][aprobada] : crear("asignatura ya aprobada")
2.6 [!cerrado][!aprobada][agotadas] : crear("convocatorias agotadas")
2.8 [!cerrado][!aprobada][!agotadas][matriculado] : crear("ya matriculado en esa asignatura")
<< L >>
:Exception
2.9 [!cerrado][!aprobada][!agotadas][!matriculado] : matricular(codAsig,nomGrupo,cursoAcademico,alu)
<< A >>
:PlanEstudios
2.9.1 : asig=buscarAsignatura(codAsig)
<< A >>
:Asignatura
2.9.2 : matricular(nomGrupo,cursoAcademico,alu)
<< L >>
asiq:Asignatura
2.9.2.1 : gru = buscarGrupo(nomGrupo,cursoAcademico)
<< A >>
:Grupo
2.9.2.2 : matricular(alu)
<< L >>
qru:Grupo
2.9.2.2.2 [plazasLibres>0]: incluir(alu)
<< A >>
:Alumno
2.9.2.2.1 [plazasLibres=0] : crear("en ese grupo no hay plaza libres")
<< L >>
:Exception
2.9.2.2.3 [plazasLibres >0] : incluirGrupo(gru)
<< P >>
alu:Alumno
2.9.2.2.3.1 : incluir(gru)
<< A >>
:Grupo




Pedido
- total: int
- estado: string
+ cobrar(): int
+ servir(): void
+ obtenerTotal(): int




Empleado
EmpleadoAComision
EmpleadoAsalariado
Director


<<Interface>>
Interface
+ metodo1(): void
Y
X
Z

TEMA 6: Herramientas de soporte a la Orientación a Objetos

Lección 1: Patrones de diseño

1. En esta lección hemos visto patrones de diseño. Establece ahora otros ámbitos relacionados con el desarrollo de sistemas software en los que también existan patrones.
2. Reúne y analiza un conjunto amplio de definiciones de patrón de diseño. Elabora después tu propia definición de patrón de diseño.
3. Busca otras clasificaciones de los patrones de diseño diferentes a la vista en clase, esto es, aquella establecida por el "Gang of Four".
4. Analiza y discute patrones de diseño en Smalltalk y Java.
5. Analiza y discute patrones de diseño para la Web.
6. **Aporta varios ejemplos (distintos a los vistos en la asignatura) donde sería útil aplicar el patrón Singleton**. Describe uno de ellos de forma completa.
7. Implementa el código del patrón Singleton en Smalltalk.
8. Simplifica el código del patrón Singleton en Java.
9. **Modifica el código del patrón Singleton en Java para permitir cinco instancias (en lugar de sólo una)**.

10. **Aporta varios ejemplos (distintos a los vistos en la asignatura) donde sería útil aplicar el patrón Facade**.
11. Utiliza el patrón Facade para implementar en Java un videoclub con las películas organizadas en tres estanterías: "novedades", "clásicos" y "el montón".
12. Implementa el ejercicio 11 en Smalltalk.

13. **Aporta varios ejemplos (distintos a los vistos en la asignatura) donde sería útil aplicar el patrón Flyweight**. Describe uno de ellos de forma completa.
14. Utiliza el patrón Flyweight para implementar en Java un procesador de palabras donde los caracteres de texto son objetos.
15. Implemente el ejercicio 14 en Smalltalk.
16. Estudia como modificar el patrón Flyweight para permitir pesos ligeros no compartidos, y justifica la utilidad de hacer esto.

17. **Aporta varios ejemplos (distintos a los vistos en la asignatura) donde sería útil aplicar el patrón State**. Describe uno de ellos de forma completa.

18. Utiliza el patrón State para implementar en Java una alarma en sus diferentes estados: activada, desactivada, sonando, en configuración, etc.
19. Implementa el ejercicio 18 en Smalltalk.
20. Modifica y amplía el diagrama de clases UML del patrón Flyweight para reflejar los cambios introducidos en el ejercicio 16.
21. Busca y analiza los diagramas de clases UML de otros patrones de diseño orientados a objetos.

AUTO-EVALUACIÓN			
Tema 6. Lección 1. PATRONES DE DISEÑO			
Fecha:		**Nota:**	

Responda V (verdadero) o F (falso) junto a cada afirmación.

1	El concepto de patrón definido por el arquitecto C. Alexander es anterior a 1970.	
2	El primer lenguaje de programación donde se aplicó el concepto de patrón de diseño fue Smalltalk.	
3	Existen patrones de diseño especializados en la Web.	
4	Un patrón define una solución exacta para un problema exacto.	
5	El uso de patrones de diseño puede reducir los tiempos de desarrollo software.	
6	Una ventaja importante de los patrones de diseño es que establecen un vocabulario de diseño.	
7	Los patrones de diseño son reutilizables, pero no favorecen la reutilización de código.	
8	Un patrón puede tener varios nombres.	
9	El propósito del patrón explica normalmente un escenario de ejemplo para motivar el uso del patrón.	
10	La estructura de un patrón se puede describir mediante un diagrama de clases en UML.	
11	Las plantillas de especificación de un patrón son abstractas y por lo tanto no incluyen código.	
12	Los participantes en un patrón nos indican las clases y objetos que debemos crear.	
13	Los patrones de diseño son independientes entre sí.	
14	De acuerdo a su ámbito, un patrón de diseño puede ser: de creación, estructural o de comportamiento.	
15	En el catálogo de patrones de "Gang of Four" hay más patrones de objetos que de clases.	
16	El patrón Iterator es de creación de objetos.	
17	El patrón Singleton es de creación de clases.	
18	El patrón Proxy es un patrón estructural.	
19	En cada problema sólo hay un patrón aplicable.	
20	La solución propuesta en un patrón delimita completamente la implementación de las clases involucradas en el mismo.	
21	Si en un patrón una clase se llama Singleton, es más conveniente utilizar ese mismo nombre para crear nuestra clase que uno más específico del problema a resolver.	

22	El patrón Singleton evita usar variables globales para acceder a un recurso único.	
23	El patrón Singleton implica que el constructor de la clase sea privado.	
24	El patrón Facade es muy apropiado para implementar el acceso controlado a un recurso único, como por ejemplo, una cola de impresión.	
25	El patrón Facade puede ser usado para simplificar y unificar las cuatro fases del proceso de compilación de código en un compilador.	
26	Siempre que se utiliza el patrón Facade las clases internas del subsistema no pueden ser accedidas directamente desde fuera.	
27	El patrón Facade facilita la división en capas de una aplicación.	
28	El patrón Flyweight y el patrón Facade pertenecen a categorías diferentes en el catálogo de "Gang of Four".	
29	El patrón Flyweight es aplicable cuando se requiere un gran número de objetos con una parte común entre sí.	
30	Para que aplicar el patrón Flyweight suponga el mayor ahorro de espacio posible es deseable que la parte intrínseca del estado de los objetos ligeros concretos sea lo mayor posible.	
31	El patrón State puede ser usado para simular que un objeto cambia de estado en tiempo de ejecución.	
32	El patrón State requiere implementar un participante que mantenga el contexto actual del objeto.	
33	El patrón State determina que el protocolo de peticiones que se pueden responder en un estado debe ser declarado en una interfaz que implementen todos los estados concretos.	
34	El patrón State es flexible en relación a dónde se implementan las transiciones entre los estados.	
35	El patrón State puede ser aplicado para implementar una conexión TCP.	

Solución:
1F 2V 3V 4F 5V 6V 7F 8V 9F 10V 11F 12V 13F 14F 15V 16F 17F 18V 19F 20F 21F 22V 23V 24F 25V 26F 27V 28F 29V 30V 31V 32V 33V 34V 35V