Relación de problemas (PDOO): 1 **Curso: 2011-12**

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?

Relación de problemas (PDOO): 1 Curso: 2011-12

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.

Relación de problemas (PDOO): 1 Curso: 2011-12

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.

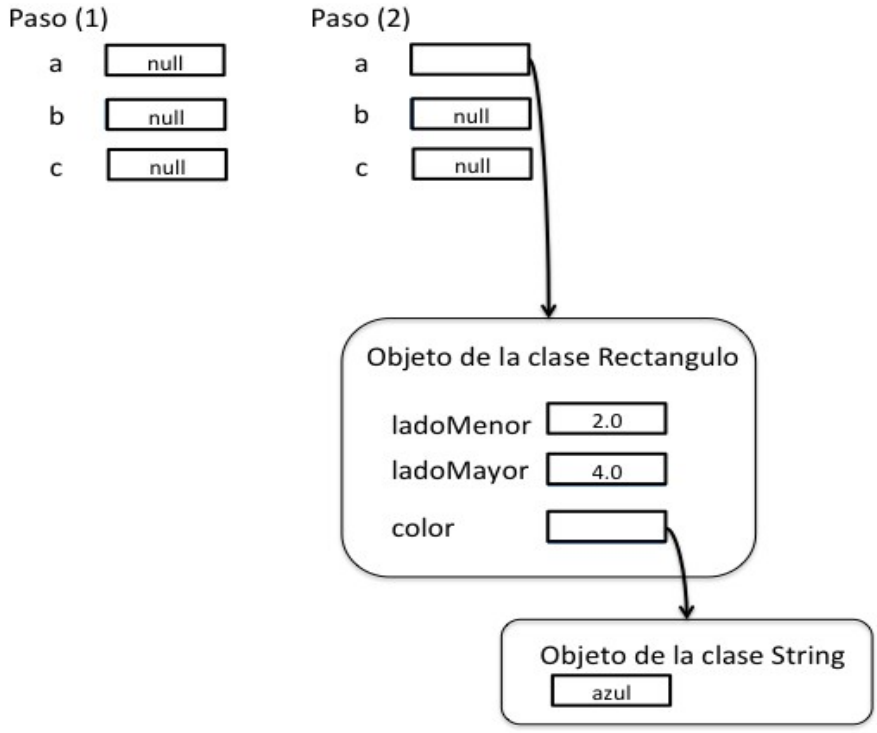
TEMA 2: Clases y métodos

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:

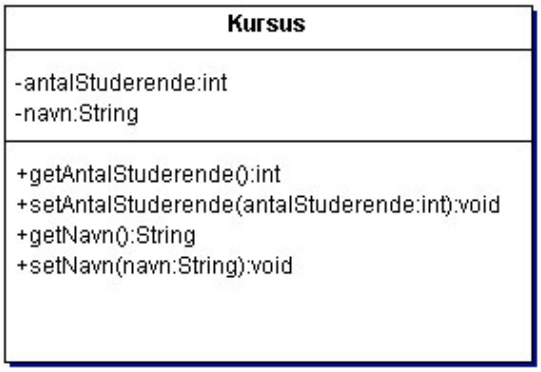
```
45 factorial.
'hola' outputToPrinter.
3+4.
var at:3.
var at:3 put:$x.
```

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.

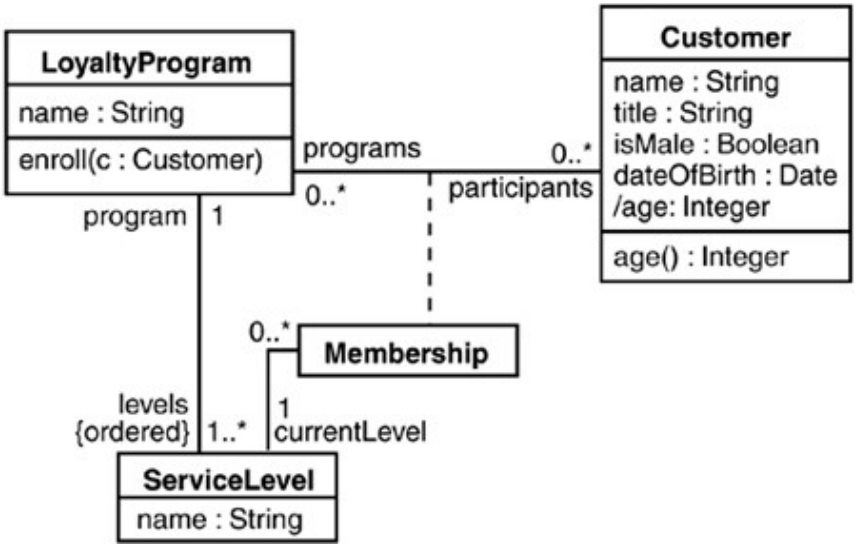
Ejercicio 10. ¿Qué mecanismos tienen Java y Smalltalk para ocultación de información?

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).

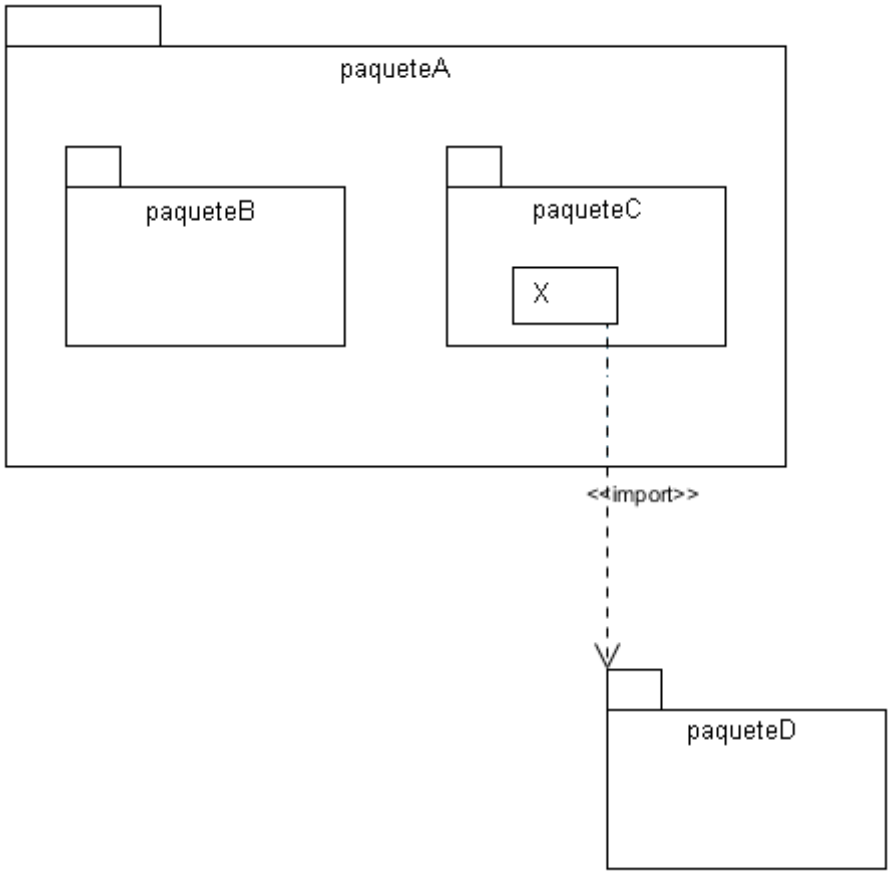


Ejercicio 13: Escribe el código Java y Smalltalk correspondiente a las clases que aparecen en el siguiente diagrama de clases sin olvidar los atributos de referencia y haciendo uso del nombre de los roles que figuran en el diagrama.



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:

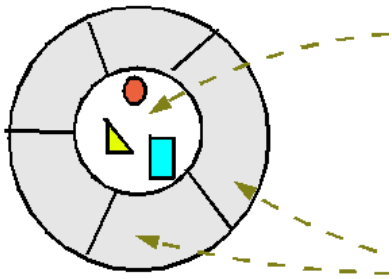
- Una clase delpaqueteB
- Una clase de paqueteA que no esté en paqueteB ni en paqueteC
- Una clase de paqueteD

c) ¿Y si se declarase “protected”?

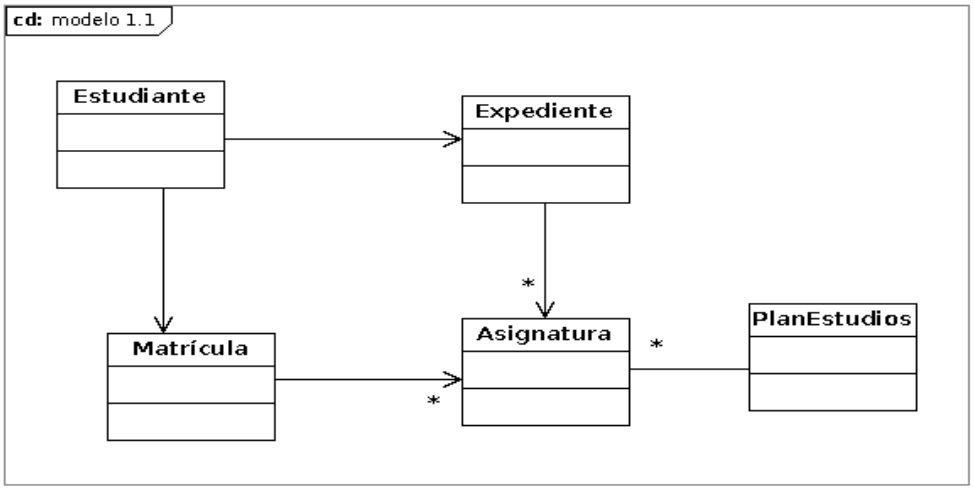
TEMA 3: Objetos y mensajes

Lección 1

1. Sabiendo que la siguiente imagen representa un objeto, ¿Qué puede simbolizar cada una de las partes señaladas?



2. Si fueses un objeto de la clase Estudiante: ¿cuál sería tu estado en este momento? ¿cuáles serían tus responsabilidades? ¿y tu identidad?
3. ¿Qué encapsula un objeto? De todo lo que encapsula, ¿qué se recomienda que esté oculto?
4. Uno de los mecanismos para copiar estado es usar el constructor de copia ¿hay que usarlo siempre que necesitemos una nueva refecencia a un objeto que ya esté creado?
5. Si la asignación "=" en un lenguaje copia identidad ¿qué ocurre con tras las siguientes instrucciones: `y=construirObjetoClaseA();  x = construirObjetoClaseA(y);` ? y ¿si la asignación copiase estado?.
6. ¿Por qué la siguiente afirmación: "El estado de un objeto viene definido por el valor que toman sus atributos básicos" es falsa?
7. A partir del siguiente diagrama de clases, obtén un esquema en el que se muestren objetos y sus enlaces (relaciones) en el siguiente supuesto: 2 objetos PlanEstudios, 6 objetos Asignatura, 5 objetos Estudiante. Haz las suposiciones que consideres necesarias.



8. ¿Qué objetos intervienen en un envío de mensaje?
9. ¿Cual es la diferencia esencial entre un envío de mensaje y una llamada a función?
10. ¿Qué es más eficiente, la ligadura dinámica o la estática? ¿por qué?

Lección 2

11. Reescribe correctamente en la segunda columna las instrucciones que dan errores de compilación en las siguientes declaraciones o asignaciones con conversiones de valores primitivos en Java:

`int a=45; long b; byte c=2; float d=2.1; char e='f'; double f=5.1;`	
`b=a;`	
`a=(long) b;`	
`d=(float) f;`	
`f=(double) a;`	
`f=(float) a;`	
`f=(int)e;`	

12. Reescribe correctamente las instrucciones en Java/C++ de declaración de arrays para que no den error de compilación:

	Java	C++
`int [][] c;`		
`int [5] e;`		
`int d[];`		
`int f[]=new int[3];`		
`int *s=new int [4];`		
`int[]x=new int[10];`		
`Array.newInstance(int, 5);`		
`int [] dims={2,4};` `Array.newInstance(Alumno, dims);`		

13. Considerando la variables declaradas en la pregunta anterior, indica qué devuelven las siguientes expresiones de envío de mensajes:

`x.getClass()`	
`x.getClass().equals(d.getClass())`	
`x.getClass()==d.getClass()`	

14. Indica a continuación si hay errores de compilación o ejecución en las siguientes instrucciones relacionadas con el uso de las clases envoltorio de Java:

`Integer i=Integer.parseInt("5");`	
`Integer i=Integer.valueOf("h12");`	
`float f=Float.floatValue();`	
`Float f2=new Float(3);` `f=f2.floatValue();`	
`double d=Double.doubleValue(3.4);`	

15. Define una clase en Java denominada Calculos que contenga:

- Un método estático que resuelva ecuaciones de segundo grado, es decir, que devuelva las

dos posibles soluciones de la ecuación ax2+bx+c=0, sabiendo que sus soluciones son:

$$x= -b \pm (a^2-4ac)^{½} /2a$$

Si no tiene solución que debe devolver null

- El método main() en el que se pruebe el método anterior con los siguientes valores, comprobando que funciona:
  - a=4, b=1, c=-6
  - a=4, b=1, c= 6
  - a=0, b=3, c= -1
  - …

16. Añade a la clase anterior dos métodos estáticos (uno recursivo y otro iteravito) que calculen el factorial de un número.
    - La versión recursiva consiste en un método que se invoca a sí mismo hasta completar el resultado:

      factorialRec(n) = n · factorialRec(n – 1)

      Cuando n sea 0 se devuelve 1 y se termina la recursividad.

    - La versión iterativa (en el método factorialIter()) consiste en realizar un bucle que vaya acumulando el resultado.

Comprueba que los programas devuelven el mismo resultado, añadiendo el código necesario para hacer tal comprobación en el método main.

17. A partir del siguiente código responde a las siguientes cuestiones:
    - ¿Cual es la funcionalidad general de este código?
    - ¿Qué se predende mostrar con él?
    - ¿Qué hacen las instrucciones 10, 15, 20, 21 y 22?
    - Explícalo paso a paso.

```
 1  // Java code
 2   import java.io.*;
 3   import java.util.*;
 4   public class Sum {
 5       public static void main (String [] args) {
 6         ArrayList listOne = new ArrayList();
 7         ArrayList listTwo = new ArrayList();
 8         ArrayList results = new ArrayList();
 9
10         int s = (args.length >  0)? Integer.parseInt(args[0]) : 5;
11
12         for (int i = 1; i < s; i++) {
13           System.out.println("  - Storing (" + i + ")");
14           listOne.add(new Integer(i));
15           listTwo.add(new Integer(2*((Integer)listOne.get(i-1)).intValue()));
16         }
17
18         for(int i = 0; i < listOne.size(); i++) {
19           Integer first = (Integer) listOne.get(i);
20           Integer second = (Integer) listTwo.get(i);
21           int sum = first.intValue()+second.intValue();
22           results.add (new Integer(sum));
23         }
```

```
24
25        for (int i = 0; i < results.size(); i++) {
26            System.out.println("  - results(" + i + ") = " + results.get(i));
27        }
28    }
39  }
```

18. Dado el siguiente código:

- ¿Qué hace?
- ¿Por qué usamos la clase Integer en lugar de int como tipo del value del Map?
- ¿Qué es lo que se hace en la sentencia situada en las líneas 6 y 7? ¿Por qué se hace así?
- ¿Qué conseguimos con el tipo Map.Entry? ¿Cuándo de usa?
- Ejecútalo, para ello investiga cómo introducir argumentos en la llamada al main().
- Haz las modificaciones necesarias para que el resultado aparezca ordenado alfabéticamente.

```
1 public class PruebaMap {
2 public static void main(String [] ar){
3  Map<String, Integer> misReferencias = new HashMap();
4   for (String termino : ar) {
5     if(misReferencias.containsKey(termino))
6       misReferencias.put(termino, new
7           Integer(Integer.valueOf(misReferencias.get(termino).intValue()+1)));
8     else
9       misReferencias.put(termino, new Integer(1));
10   }

11  for (Iterator itMap = misReferencias.entrySet().iterator();itMap.hasNext();)
12  {
13    Map.Entry ele=  (Map.Entry)itMap.next();
14    System.out.println("Termino : " + ele.getKey() + " numero de veces : "
15          + ele.getValue());

16  }
17 }
18}
```

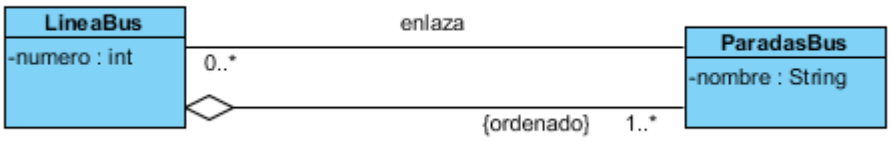
19. Necesitamos ordenar una pila de ruedas para que no se caigan, para ello hay que poner primero las de mayor diámetro y a igualdad de diámetro mayor grosor. A partir del siguiente código de la clase Rueda, implemeta el método compareTo() y escribe un método main() donde se vea cómo se ordena una colección de ruedas. Haz esta prueba con todas las clases que implementan la interface List que no sean abstractas.

```
public class Rueda implements Comparable{
    private double diametro;
    private double grosor;

    public Rueda(double unDiametro, double unGrosor){
        diametro = unDiametro;
        grosor = unGrosor;
    }
    public double getDiametro(){
        return diametro;
    }
    public double getGrosor(){
```

```
        return grosor;
    }

    @Override
    public int compareTo(Object o) {
       // implementación que se pide
    }
}
```

20. Usando el ejemplo anterior, define un TreeSet para almacenar las ruedas, usando como definición de la variable la interface SortedSet. Busca en la clase Collections un método que ordene los elementos de un TreeSet. Verifica con un ejemplo que se ordena un conjunto de ruedas y que no hay elementos repetidos en esa colección. Ten en cuenta que si queremos que no haya en el conjunto dos ruedas con el mismo diámetro y grosor es necesario redefinir en la clase Rueda el metodo equals().

21. Implementa una clase LineaBus definida por un número y por las paradas de autobús que la forman. Cada parada de autobús tiene un nombre y una lista líneas con las que enlaza (tal y como se indica en la figura). Implementa un main() en el que se muestre la ruta (lista de paradas) que tiene que hacer el autobús durante 10 minutos, sabiendo que el autobús tarda 1 minuto en ir de una parada a otra y permanece medio minuto en cada parada, mostrando asimismo los números de las líenas con las que enlaza en cada una de esas paradas. Presta atención a que las paradas de autobús de una línea van ordenadas, comenzando por la parada de cabecera y terminando en la parada de finalización.



22. Partiendo del diagrama anterior, define una nueva clase que represente el servicio de transporte público en autobús, y que tenga métodos para hacer las siguientes consultas:
    1. Mostrar todas las líneas con sus paradas y con los número de líneas con los que enlaza.
    2. Estoy en una parada de autobús y quiero ir a otra parada ¿qué línea tengo que coger?
    3. Incluir el tratamiento de las excepciones en los siguientes casos:
        - Cuando se proporcione un número de línea inexistente.
        - Cuando se proporcione una parada inexistente.
        - Cuando se quiera ir a una parada y no haya comunicación.
        - Todas las demás excepciones que consideres necesarias.

23. Indica el tipo de las siguientes expresiones en Smalltalk:

`$3`	
`Smalltalk`	
`#(3 'hola')`	
`#MiClase`	
`true`	
`self`	
`super`	
`[a:= 3]`	

`45 factorial.`	
`var`	
`'a' factorial.`	

24. Pone ejemplos de uso de las pseudovariables constantes en Smalltalk.

25. Observa el siguiente código e indica el ámbito de cada una de las variables:

```
Object subclass: #Empresa
instanceVariableNames: 'cif empleados'
classVariableNames:'PorcentajeRetencion'
poolDictionaries:'DatosFiscales'

Smalltalk at:#DatosFiscales put:Dictionary new.
DatosFiscales at:#iva put:0.07

"un método"
nombre:unNombre
      |var2|
       var2 := nombre
       ^var2
```

26. Indica el resultado de cada expresión de mensaje:

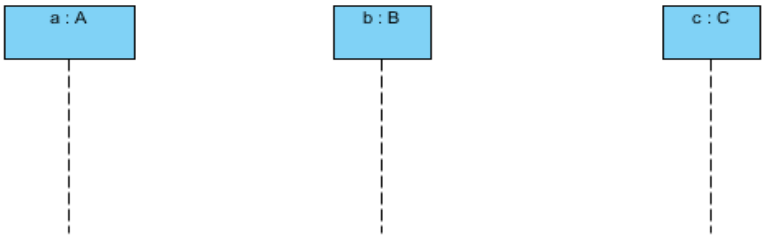
`4 + 7 * 5 + 2`	
`10 raisedTo: 2 + 4 sqrt`	
`edades at: 'Juan' put: (edades at:'Pedro')`	
`edades at:'Juan' put: edades at:'Pedro'`	
`vector at: 1 put: 7 ;  at: 2 put: 45 ; at: 3 put: 23`	
`[unArray at:1 put:3]`	
`[unArray at:1 put:3] value`	
`[:a :b | (a,b) size] value:'hola' value:'amigo'`	
`[:a  | a:=a*2] value: 7`	

27. Escribe un método en Smalltalk que cuente el número de cifras del receptor (un entero hasta 999). Así, para 234 la salida debe ser "3 cifras". Para cero la salida es "1 cifra".
28. Implementa un método en Smalltalk que encuentre el primer entero que haga que la suma de la serie 1, 2, 3, 4, 5, 6, 7, 8, 9,...; supere 30.
29. Implementa el método `whileTrue:` de manera recursiva en Smalltalk.
30. Implementa un método en Smalltalk que ejecute un bloque de código un determinado número de veces.
31. Implementa un método en Smalltalk que devuelva el primer elemento de un Array que sea mayor que el valor entero que se pasa por argumento.
32. Implementa un método en Smalltalk que devuelva todos los elementos de un Array mayores que el que se pasa como argumento.
33. Implementa un método en Smalltalk que devuelva el factorial de los elementos de un Array de enteros.

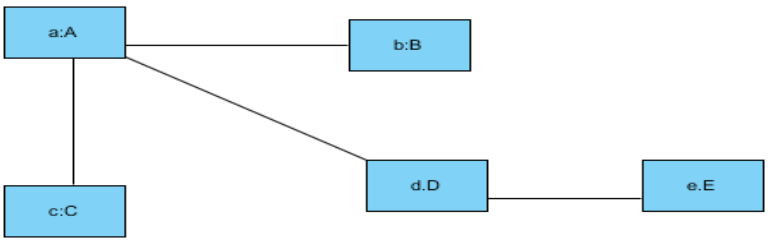
34. Implementa un método en Smalltalk que indique si un valor se encuentra en un Array, usando el método `do:`
35. Implementa en Smalltalk el método `do:`, usando el método `whileTrue:`
36. Implementa en Smalltalk el método `detect:`, usando el método `do:`
37. Implementa en Smalltalk el método `collect:`, usando el método `do:`
38. Implementa en Smalltalk los ejercicios del 18 al 22.

Lección 3

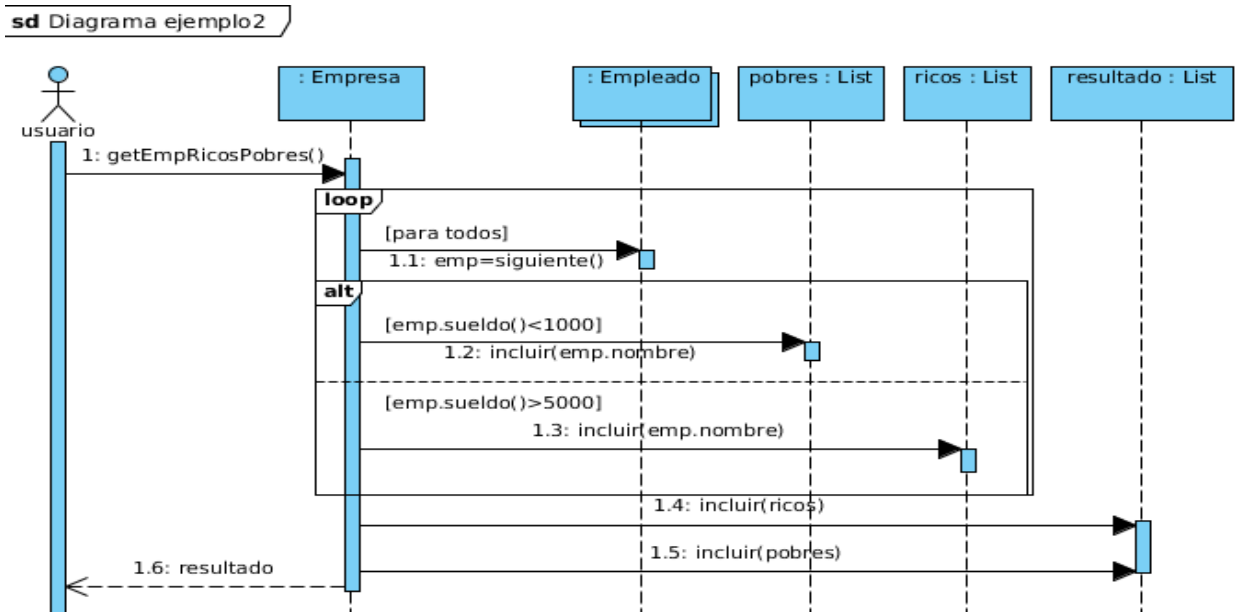
39. En general, ¿qué representan los diagramas de interacción de UML?, ¿cuáles son sus componentes principales?
40. Establece la correspondencia entre los diagramas de secuencia y los diagramas de comunicación, indicando cómo se representan los siguientes elementos: objetos, mensajes, canal de comunicación, estructuras de control, subordinación en el envío de mensaje y orden de un mensaje en una secuencia de mensajes.
41. ¿Qué relación existe entre el diagrama de clases y los diagramas de interacción?
42. 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.



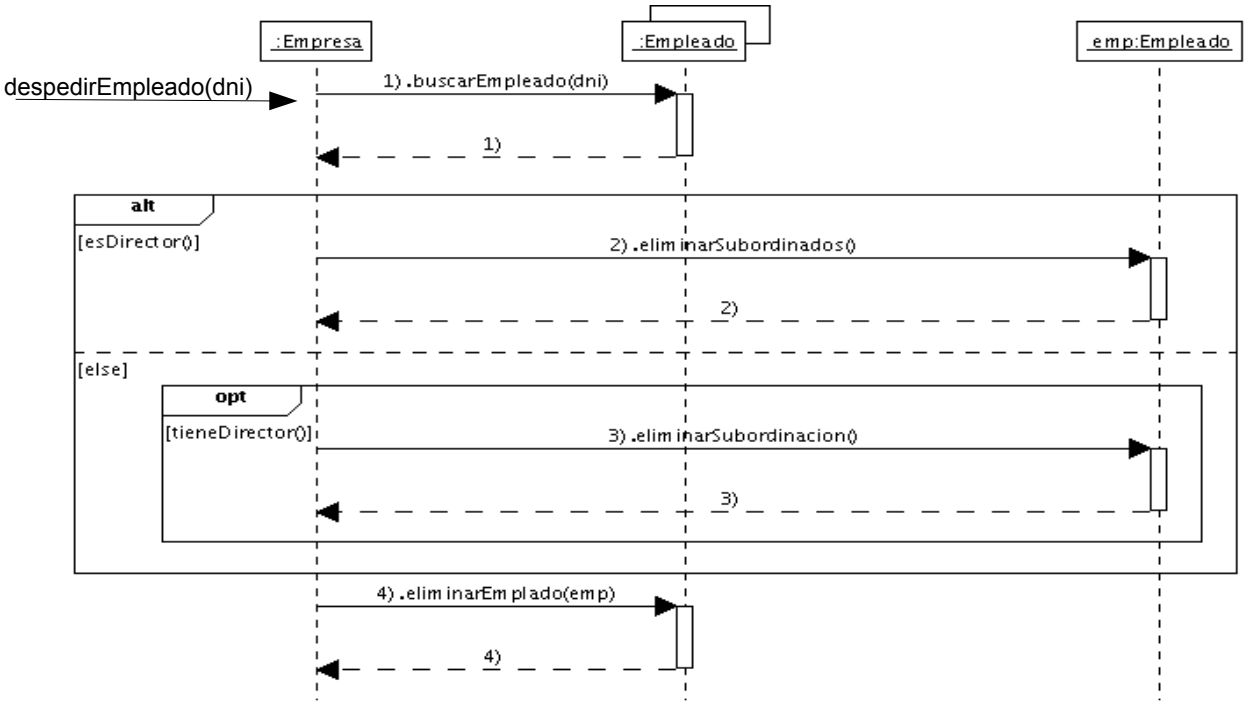
43. A partir del siguiente esquema elabora un diagrama de comunicación genérico en el que haya al menos una estructura de control iterativa y otra selectiva. Tradúcelo a su correspondiente diagrama de secuencia e impleméntalo en Java.



44. A partir del siguiente diagrama de secuencia, obtén su diagrama de comunicación equivalente.

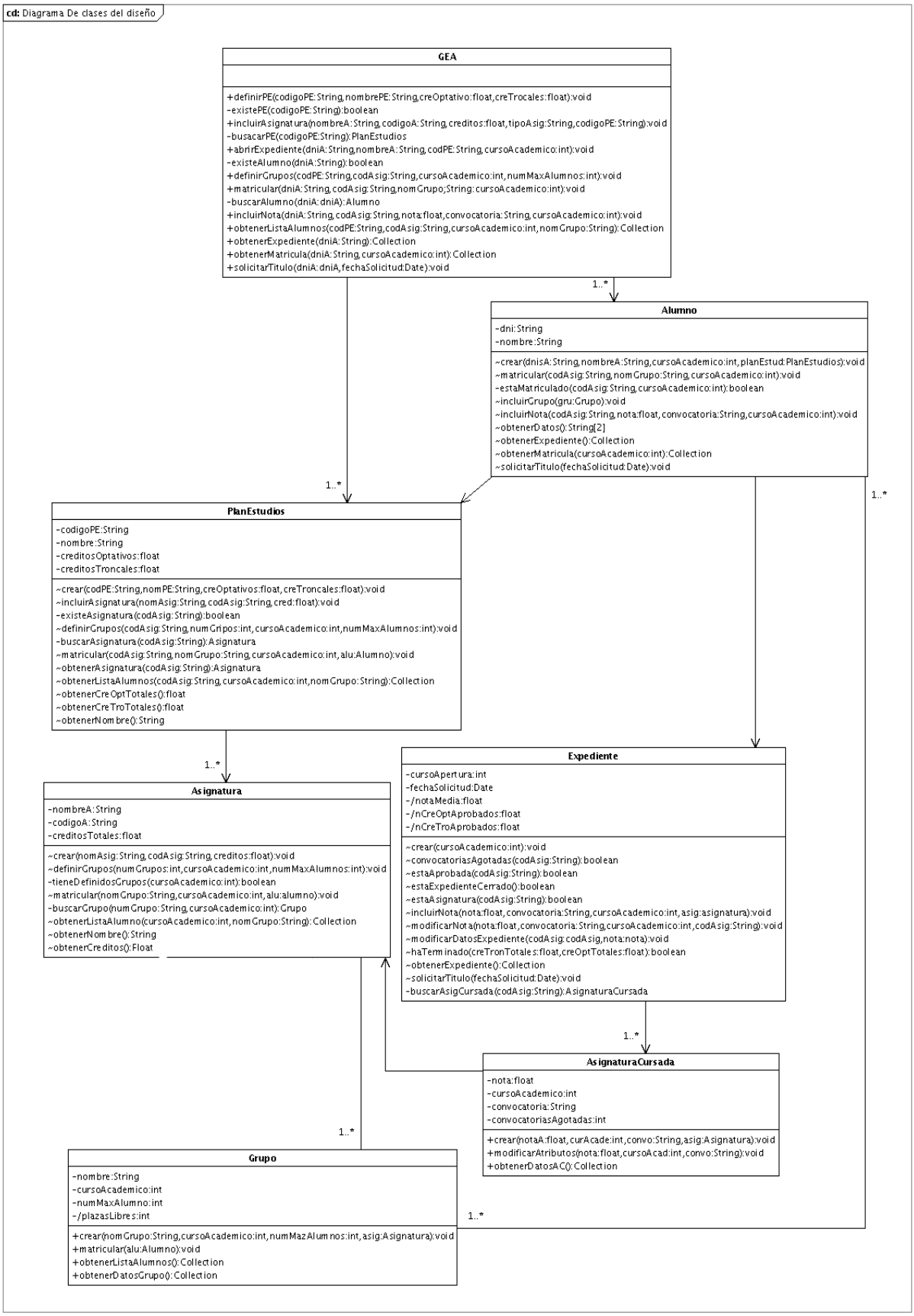


45. A partir del siguiente diagrama de secuencia, obtén su diagrama de comunicación equivalente e impleméntalo en Java.

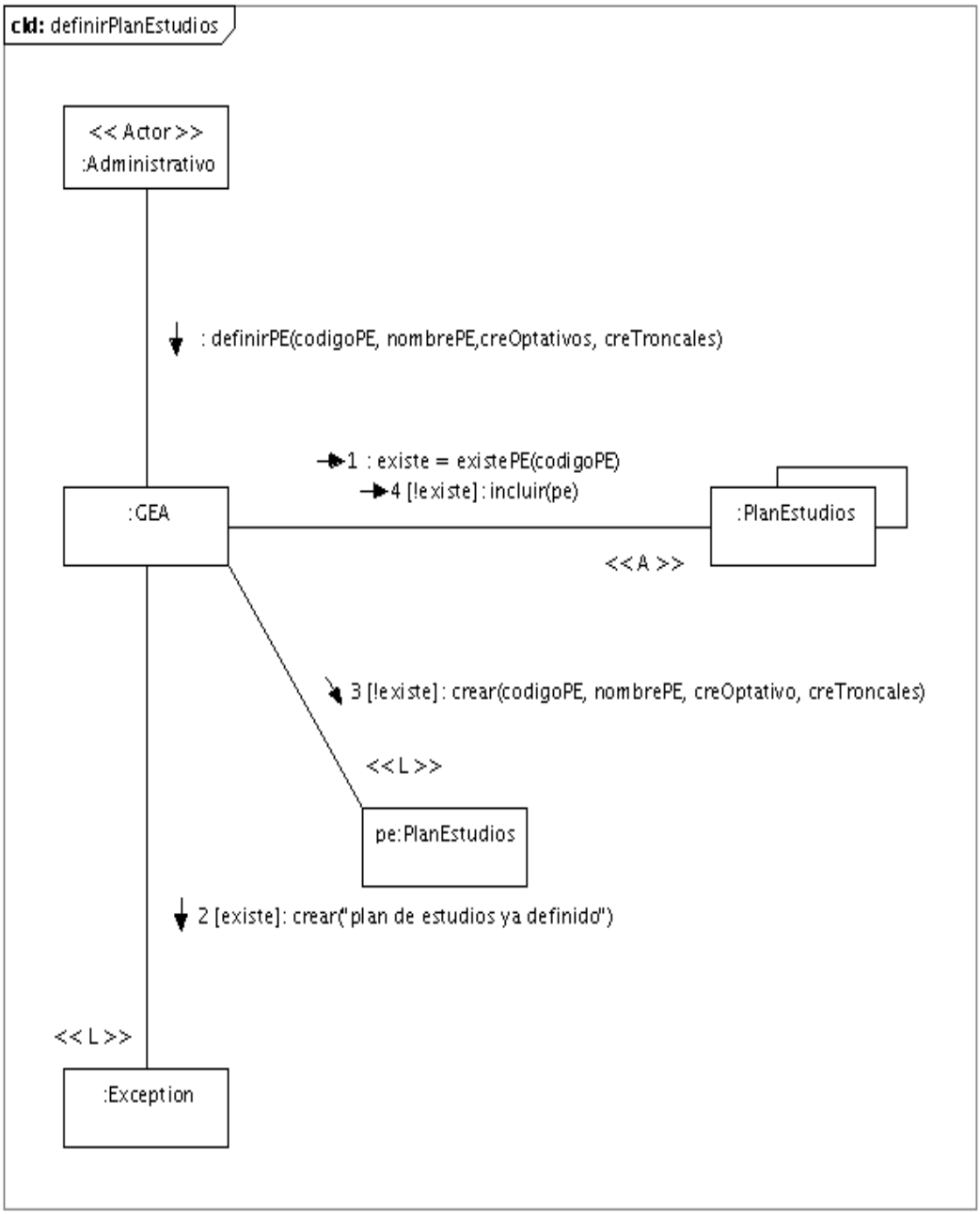


46. A partir del código desarrollado en los ejercicios 21 y 22 en Java representa:
    1. Su diagrama de clases completo.
    2. Los diagramas de interacción (secuencia y/o comunicación) de las operaciones desarrolladas.

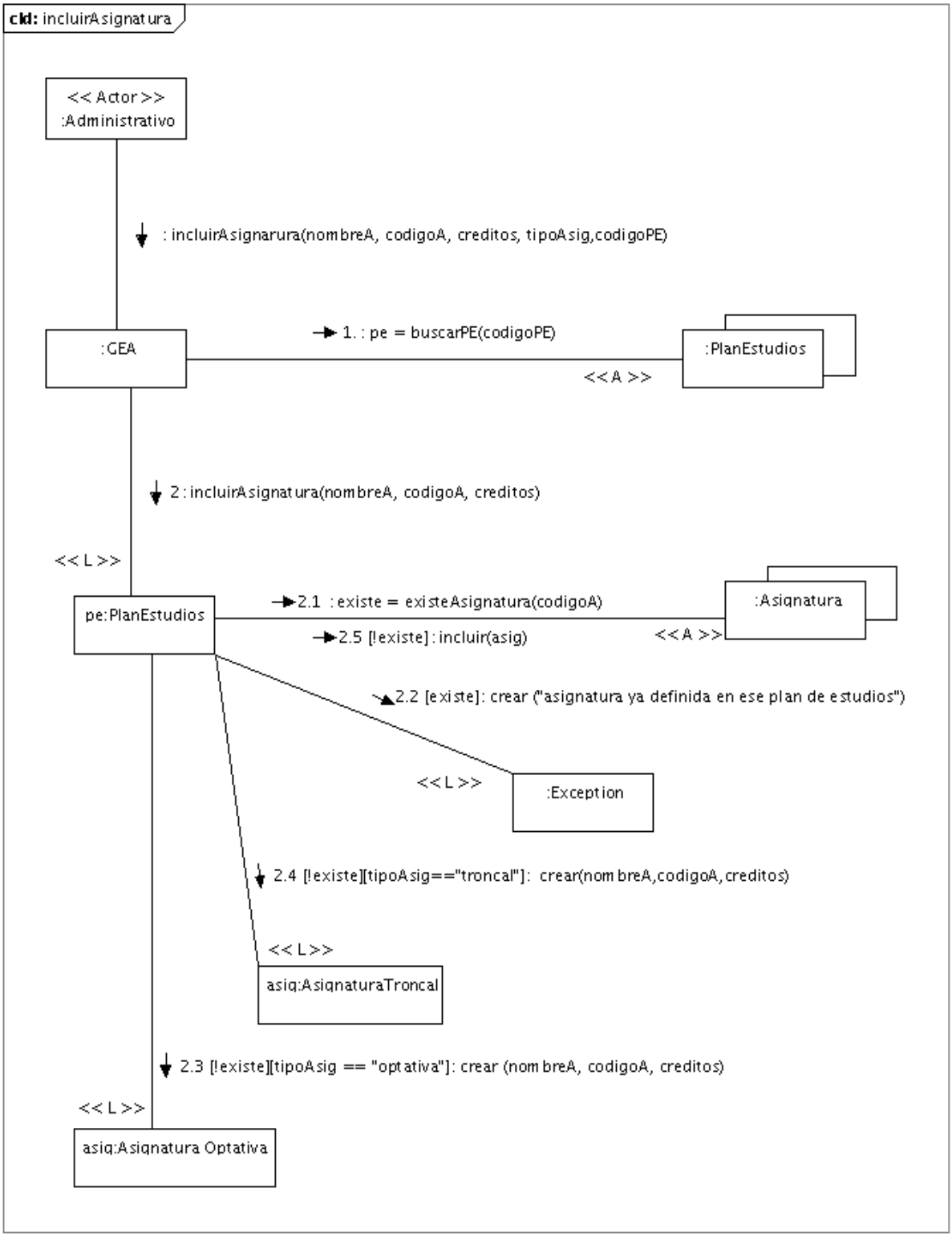
47. Implementa en Java el diagrama de clases y los 4 diagramas de comunicación que se proporcionan a continuación. Traduce al menos un diagrama de comunicación de los proporcionados a su correspondiente diagrama de secuencia.



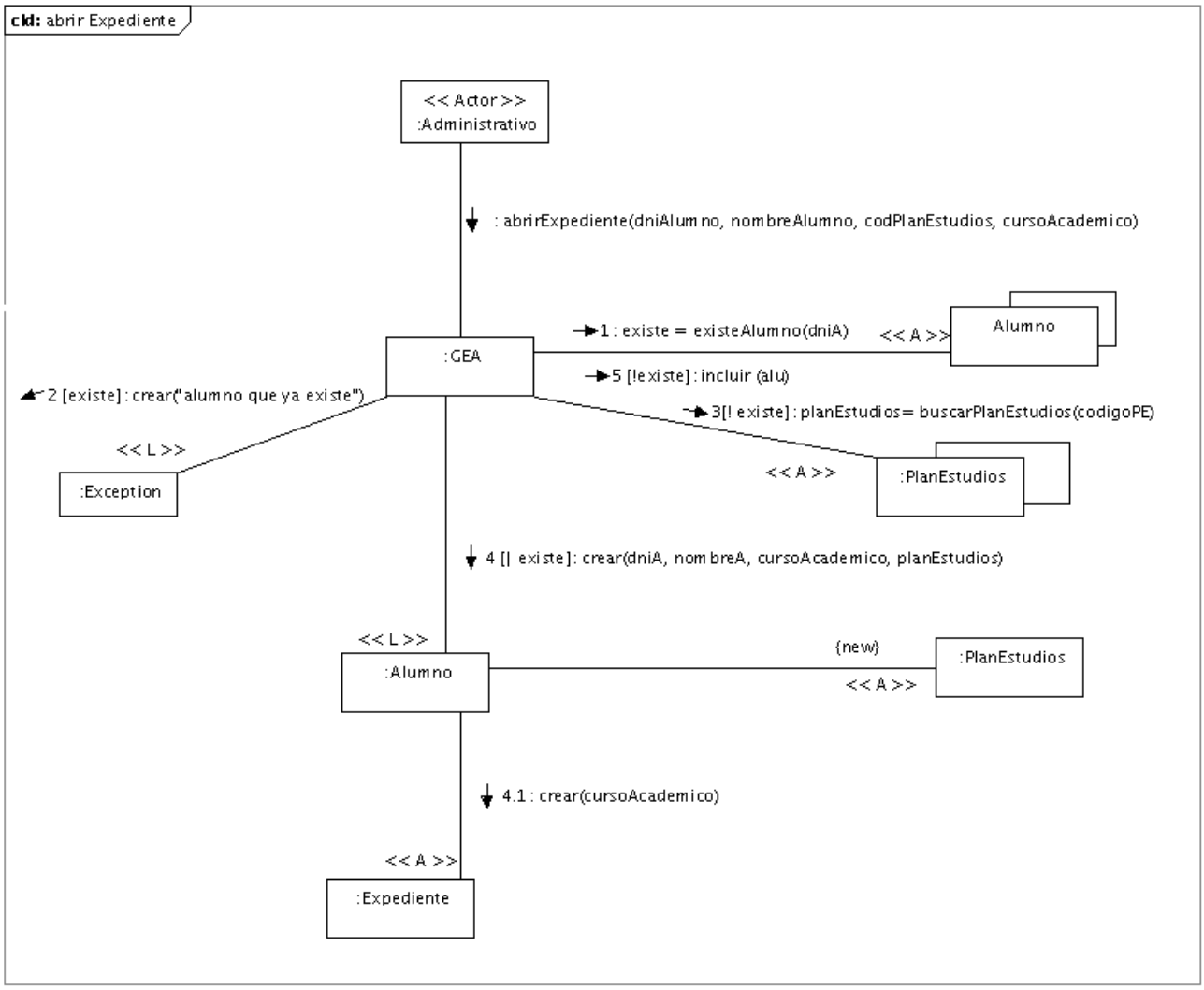
a) Operación que se encarga de definir un objeto plan de estudios, inicializarlo y almacenarlo en la colección de planes de estudio, si ya existe un objeto plan de estudios con ese código se lanza una excepción.



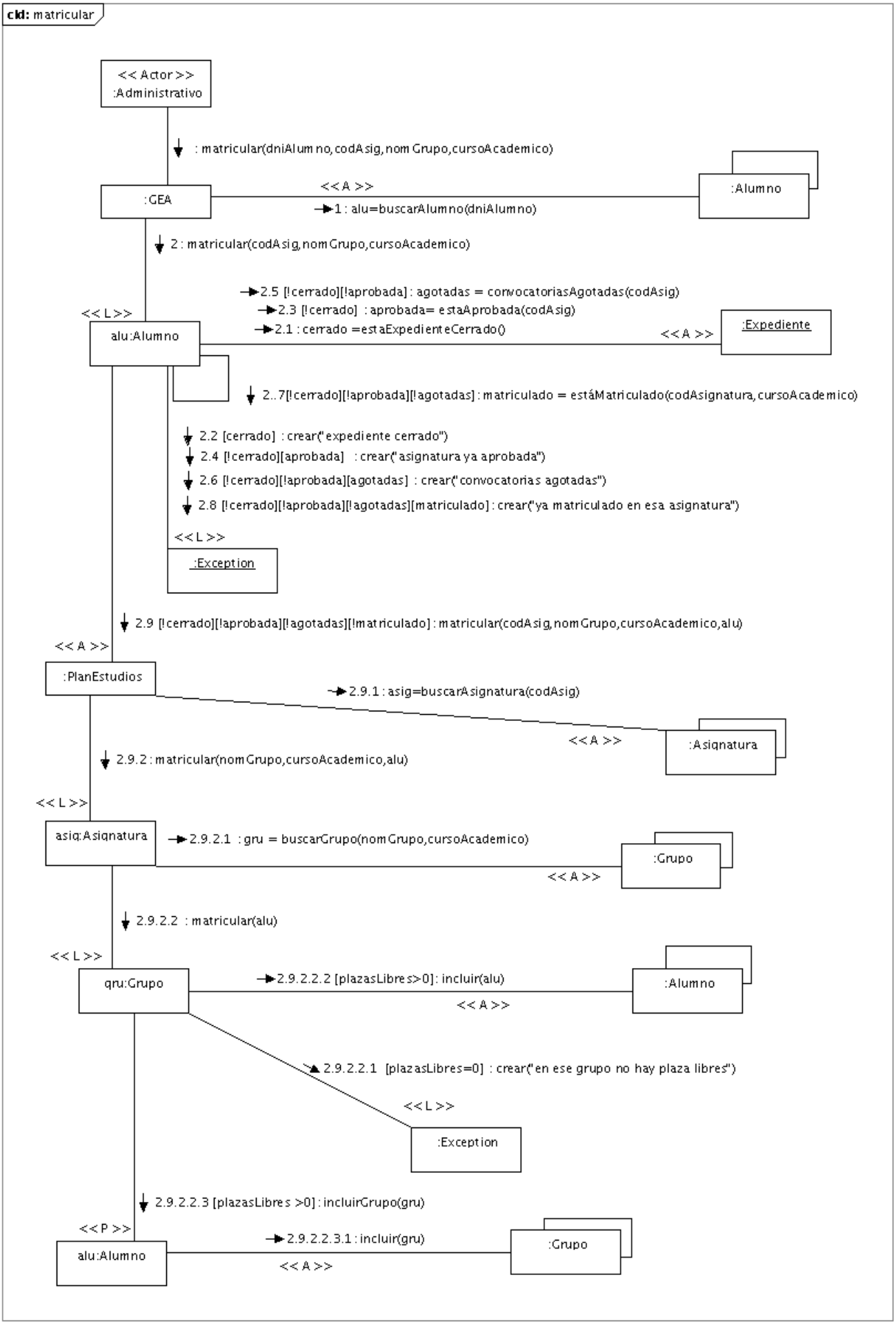
b) Operación encargada de incluir una nueva asignatura dentro de un plan de estudios determinado, si ya existe dentro de ese plan de estudios una asignatura con ese código, se lanza una excepción.



c) Operación encargada de abrir expediente a un alumno en un plan de estudios, se lanza una excepción si el alumno ya tiene abierto un expediente.



d) Operación encargada de matricular a un alumno en una asignatura. Debes comprobar que el alumno tenga el expediente abierto, que la asignatura no esté aprobada, que no tenga las convocatorias agotadas o que el alumno ya esté matriculado en esa asignatura.


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 >>
asig:Asignatura
2.9.2.1 : gru = buscarGrupo(nomGrupo,cursoAcademico)
<< A >>
:Grupo
2.9.2.2 : matricular(alu)
<< L >>
gru: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

Relación de problemas de PDOO: Temas 4 y 5

Herencia

1. ¿Cómo afecta el especificador de acceso de una variable/método para ser heredado según sea: private, public, protected o visibilidad de paquete?
2. ¿Es posible heredar variables y métodos de clase o sólo de instancia?
3. ¿Cómo se puede acceder a las variables privadas de una superclase desde una subclase?
4. ¿Cómo se puede anular el comportamiento de un método de la clase padre en una clase hija? Pon un ejemplo en Java.
5. ¿Es posible redefinir en las subclases un método de la superclase definido como `final`?
6. Indica la diferencia entre sobrecargar y redefinir un método.
7. ¿Para qué sirve la anotación @Override en Java?
8. Explica cómo funciona la sobreescritura de métodos, es decir, cómo decide el sistema qué método invocar en cada caso.
9. ¿Qué diferencia hay entre generalización y especialización?
10. ¿Por qué no debe utilizarse la limitación para heredar?
11. ¿Se tiene un mal diseño si una clase tiene muchas subclases?
12. ¿Qué quiere decir que una clase puede estar cerrada y al mismo tiempo abierta?
13. ¿Cuántos niveles de jerarquía de herencia se pueden hacer? ¿Cuál crees que es el nivel óptimo?
14. ¿Para qué sirve una clase abstracta?
15. ¿Para qué sirve una interface o interfaz de Java?
16. ¿Se pueden crear instancias de las clases abstractas?
17. ¿En qué se diferencia a nivel conceptual una clase abstracta de una interface?
18. ¿Qué problemas presenta la herencia múltiple? Explícalos.
19. ¿Puede evitarse el uso de la herencia múltiple?
20. ¿Puede haber herencia repetida sin que haya herencia múltiple?

21. Sea una clase *A* con un método *a*, una subclase de *A*, *B*, con un método *b*, y una subclase de *B*, *C*, con un método *c* y un método *a* que redefine al heredado. Dado el método *c* que usa el método *b* y dado el método *b* que usa el método *a*. ¿Al ejecutar el método *c*, qué método *a* se ejecutaría, el de *A* o el redefinido en *C*? Piénsalo y pruébalo antes de contestar.

22. Para inicializar las variables y métodos de una clase, se puede usar *super* en el constructor de la subclase. Pon un ejemplo diferente del de las transparencias de clase que ilustre su uso en Java y en Smalltalk.

23. Añade al ejemplo de Persona-Alumno-Profesor dos subclases para Profesor: Visitante y Asociado. Visitante tendrá otras dos subclases: Extranjero y Nacional. Si es necesario, redefine los métodos “darClase” y “hablar” para que funcionen según el tipo de profesor. Indica si las superclases deben sufrir algún cambio. Impleméntalo en Java y Smalltalk, creando objetos de todas las clases, mostrando sus atributos y funcionalidad.

24. Implementa en Java la clase abstracta Instrumento y las clases Violín y Guitarra que implementan la funcionalidad propia. Indica cómo se usarían las clases.

25. ¿Cómo modificarías el ejemplo anterior para incluir también tipos de instrumentos de viento?

26. Dada la interfaz Figura y la clase FormaG, se creó la clase Rectángulo. Reproduce el ejemplo creando la clase Círculo. Añade los atributos y funcionalidad que creas conveniente.

27. ¿Cómo se representa la herencia en UML?

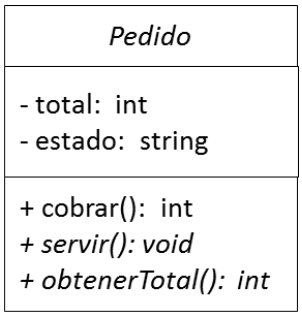
28. ¿Qué diferencia hay entre interfaces y clases abstractas en UML? ¿Cómo se modelan?

29. ¿Cómo se llama la relación entre una clase y una interfaz en UML?

30. ¿Puede existir una relación de generalización entre interfaces? Explícalo.

31. ¿Cuáles son los elementos que se heredan de una superclase en UML?

32. Convierte la siguiente clase abstracta en una clase concreta que implementa una interfaz.



33. Traduce los diagramas de la pregunta anterior a Java. ¿Se podría traducir el diagrama obtenido de la pregunta anterior a Smalltalk?

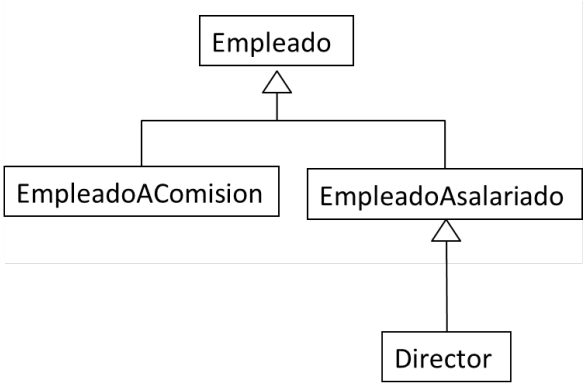
Polimorfismo

1. ¿Qué es el polimorfismo?
2. ¿Bajo qué condición una variable puede referenciar a objetos de diferentes clases?
3. Proporciona ejemplos de compatibilidad de tipos y prográmalos en Java.
4. Explica con detalle el concepto casting.
5. ¿Es posible crear referencias a interfaces?
6. ¿Se pueden crear instancias de interfaces?
7. ¿Qué diferencia hay entre la ligadura dinámica y la ligadura estática, y cuál de ellas soporta mejor el polimorfismo? Pon un ejemplo.
8. Observa el siguiente código y responde:

```
public class Poli4{

	public static void main(String[] args) throws
	javi.io.IOException{

		Persona uno;

		System.out.println("Introduzca 1 para el ladrón y 2 en
		otro caso");

		int dato = System.in.read();

		if ((char) dato =='1')

			uno= new Ladron();

		else

			uno = new Persona();

		System.out.println("Clase del objeto:" +
		(uno.getClass().getName()));

		uno.canta();

}
```

- ¿De quién tiene que heredar la clase “Ladron” para que no exista un error en tiempo de compilación?
- ¿Cómo tiene que ser la ligadura para que exista polimorfismo?
- ¿Dónde ocurre el polimorfismo?

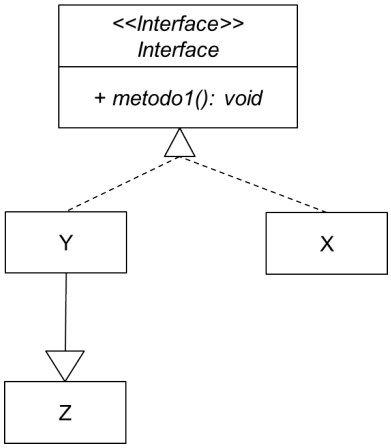
9. Dada la siguiente jerarquía de herencia:



- Indica los errores que se producen en el siguiente código:

```
EmpleadoAComision ec = new EmpleadoAComision(“Ana”, 5499, 0.3));
EmpleadoAsalarado ea = new EmpleadoAsalariado(“Juan”,65420,1200));
Director d = new Director(“Paqui”,76540,3000));

ArrayList<EmpleadoAsalariado> asalariados =
                                    new ArrayList<EmpleadoAsalariado> ();
asalariados.add(ec);
asalariados.add((EmpleadoAsalariado) ec);
asalariados.add(d);
asalariados.add((EmpleadoAsalariado) d);
Empleado e = (Empleado) d;
asalariados.add(e);
asalariados.add((EmpleadoAsalariado) e);
```

10. Dado el siguiente diagrama:



- Indica qué errores de compilación tiene el siguiente código

```
public interface Interface {
	public void metodo1();
}

public class Referencias{
	public static void main(String[] args){
		X x = new X(“nombre1”);
		Interface1 in1 = x;
		in1.método1();
		Z z = new Z(“nombre2”)
		Interfacece1 in1 = z;
		in1. Método1();
		Interface1 in2 = new Interface1();
		in2.metodo1(); }}
```

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