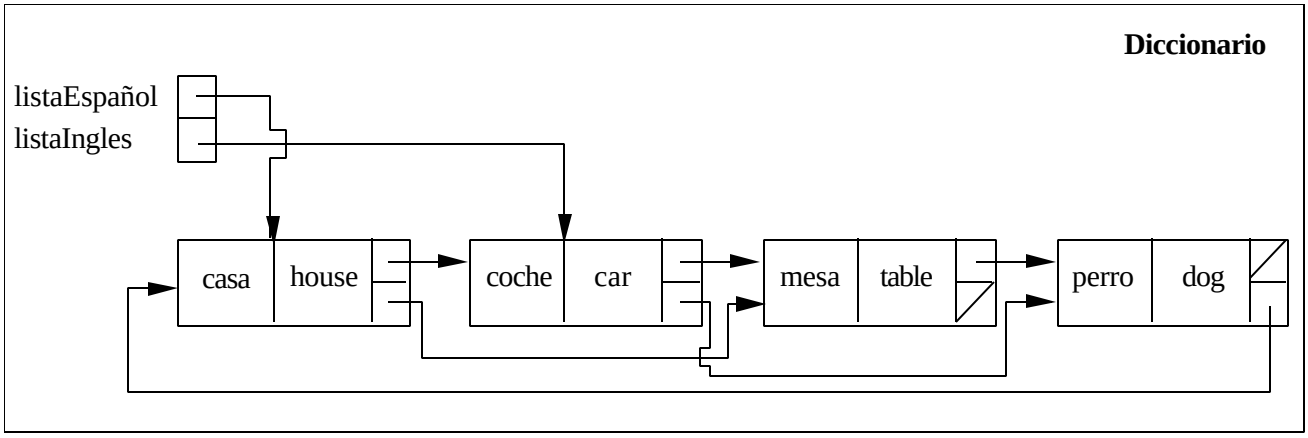
Para la resolución de todos los ejercicios de esta sección se dispone de una implementación del TAD Lista a través de las clase LinkedList, y su iterador LinkedListItr. A continuación se exponen sus definiciónes.

```
// Clase LinkedList

// CONSTRUCTOR: sin inicializador
// El acceso se realiza a través de la clase LinkedListItr
//
//******************OPERACIONES PUBLICAS*****************
// boolean isEmpty( )     --> Devuelve true si es vacia
// void emptyList( )      --> Elimina todos los elementos


// clase LinkedListItr; mantiene la "posición actual"
//
// CONSTRUCTOR: con el LinkedList a la que LinkedListItr está
// permanentemente ligada
//
// ATRIBUTOS
// private current: mantiene la “posición actual” en la lista
//******************OPERACIONES PUBLICAS*****************
// void insert(Object x)        → Inserta x tras la posición current
// void remove(Object x )       → Elimina el primer x
// void removeCurrent()         → Elimina el elemento current
// boolean find(Object x)       → Coloca current para poder ver x
// void goFirst()               → Coloca current en la primera posición
// void advance()               → Avanza current al siguiente nodo
// boolean isOnList()           → No está vacía
// Object getCurrent()          → Elemento en la posición current
// Object getPrevious()         → Elemento de la posición anterior al current
```

(J94) Disponemos de una clase de representa un diccionario bilingüe español-inglés. Dicha clase mantiene ordenada la lista de palabras tanto por un idioma como por otro.

Cada palabra en castellano, (y respectivamente, cada palabra en inglés) se encuentra en el diccionario una sola vez, por tanto, una misma palabra no puede tener dos traducciones distintas

En el siguiente ejemplo se puede observar cómo la lista española contiene las palabras *casa coche, mesa* y *perro* (por orden alfabético en lengua española) y la lista inglesa contiene las palabras *car, dog*, *house* y por último a *table* (por orden alfabético en lengua inglesa).



Se pide:

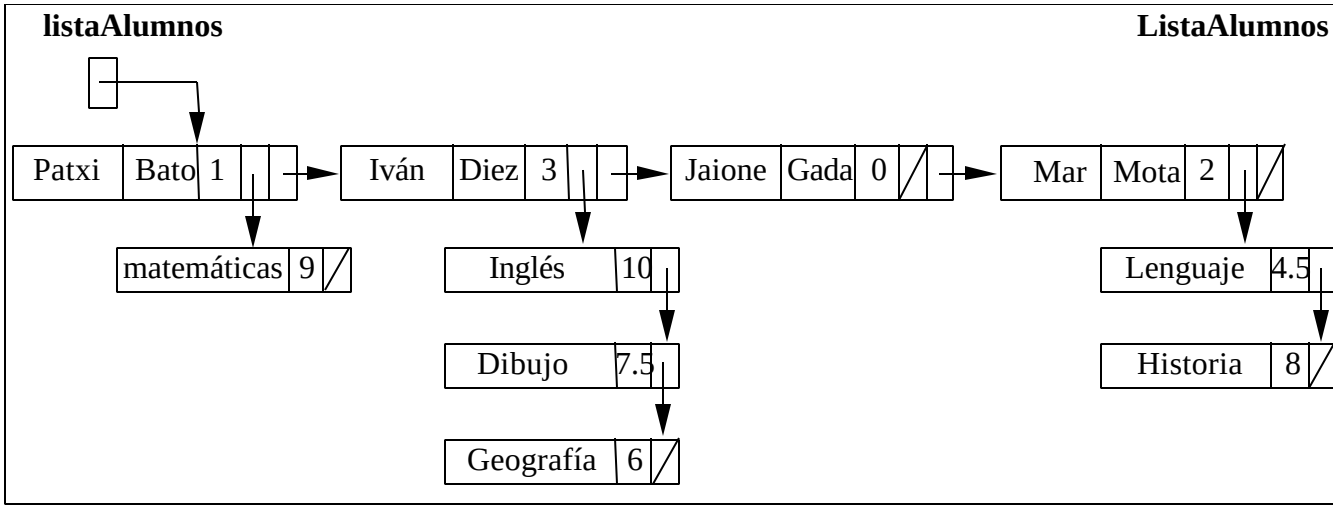
a) La(s) clase(s) necesarias para recoger la especificación definida.

b) **El diseño e implementación en Java del método de la clase Diccionario traducir.** Este método recibirá como entrada una palabra y si la palabra es inglesa o española (boolean), y visualizará por pantalla como resultado la traducción de la palabra al otro idioma: Si la palabra no está en el diccionario. En caso de no existir, lo indicará con el correspondiente mensaje.

```
public void traducir(String p, boolean b)
```

(S94) Disponemos de una clase que representa a la lista de alumnos de un grupo junto a las calificaciones obtenidas en las asignaturas en las que se han presentado. La clase está ordenada por apellido de los alumnos pero para cada alumno las asignaturas han sido colocadas sin seguir ningún orden concreto.

En la siguiente figura se muestra cómo se organiza la clase de alumnos, ordenada por apellidos únicamente.



Se pide el diseño e implementación en Java de:

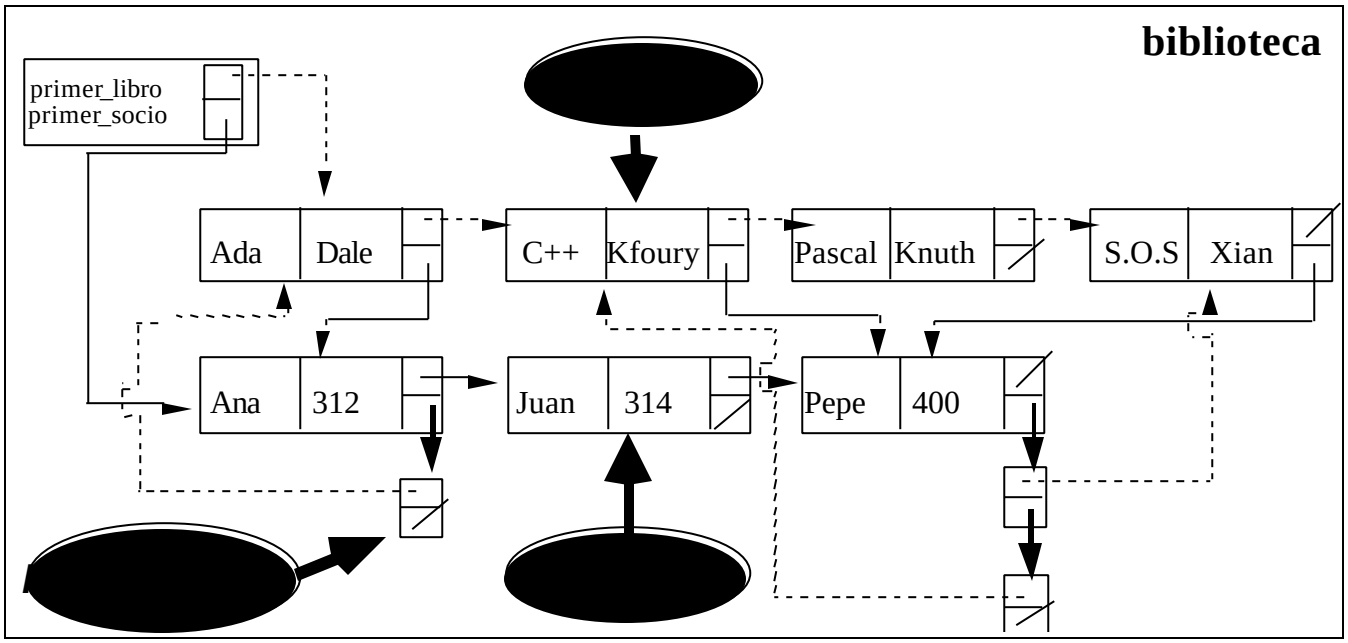
a) La(s) clase(s) necesarias para recoger la especificación definida.

b) Diseñar e implementar el Java el método de la clase ListaAlumnos visualizarResultados tal que dada el nombre de una asignatura, escriba en la salida estándar el nombre y apellido de los Alumnos de esa lista que han aprobado esa asignatura.

```
public class ListaAlumnos extends Object {
 public void visualizar_aprobados (String asign) {
 ..............
 ..............
 }
}
```

(J95) Disponemos de una clase que representa una biblioteca. Dicha clase consta de la lista de libros que hay en la biblioteca, y de la lista de socios que pertenecen a ella. Cada una de estas listas está ***ordenada***, la de socios por el ***número de carnet*** y la de libros por ***orden alfabético del título del libro***.

La figura siguiente muestra una biblioteca que dispone de 4 libros y tiene 3 socios:



Cada elemento de la **lista de libros** consta de título, autor y una variable que da acceso al socio que lo tiene en su poder, (si el libro no está prestado esa variable es null). Cada elemento de la **lista de socios** consta de nombre, número de carnet y la **lista de préstamos** del socio (libros que ha sacado de la biblioteca). Cada elemento de una lista de prestamos consta de una variable al libro en cuestión (al elemento de la lista de libros).

Se pide :

a) La(s) clase(s) necesarias para recoger la especificación definida.

b) **El diseño e implementación** en Java del método **atenderPetición** de la clase Biblioteca, tal que dado el carnet de un socio y el título de un libro atienda la petición de préstamo de la siguiente forma:

- Si el libro no estuviera en la biblioteca, escribir un mensaje en la salida estándar notificando el error (la biblioteca puede no tener libros).
- Si el libro ya estuviera prestado, escribir un mensaje en la salida estándar dando el nombre de la persona que lo tuviera.

- Si el libro no estuviera prestado, entonces actualizar la biblioteca reflejando que ese libro se va a prestar y a quién.

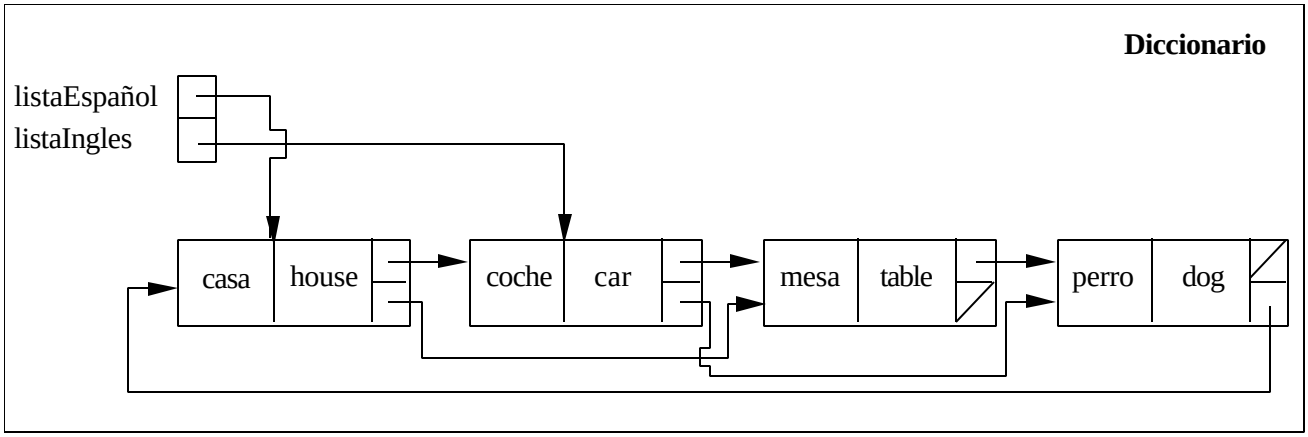
El carnet que nos dan como entrada al procedimiento siempre se encontrará en la lista de socios, por tanto, no habrá que tener en cuenta el caso en el que no estuviera.

```
public  void prestar (Socio pSoc, String pLib)
```

(S95) Disponemos de una clase de representa un diccionario bilingüe español-inglés. Dicha clase mantiene ordenada la lista de palabras tanto por un idioma como por otro.

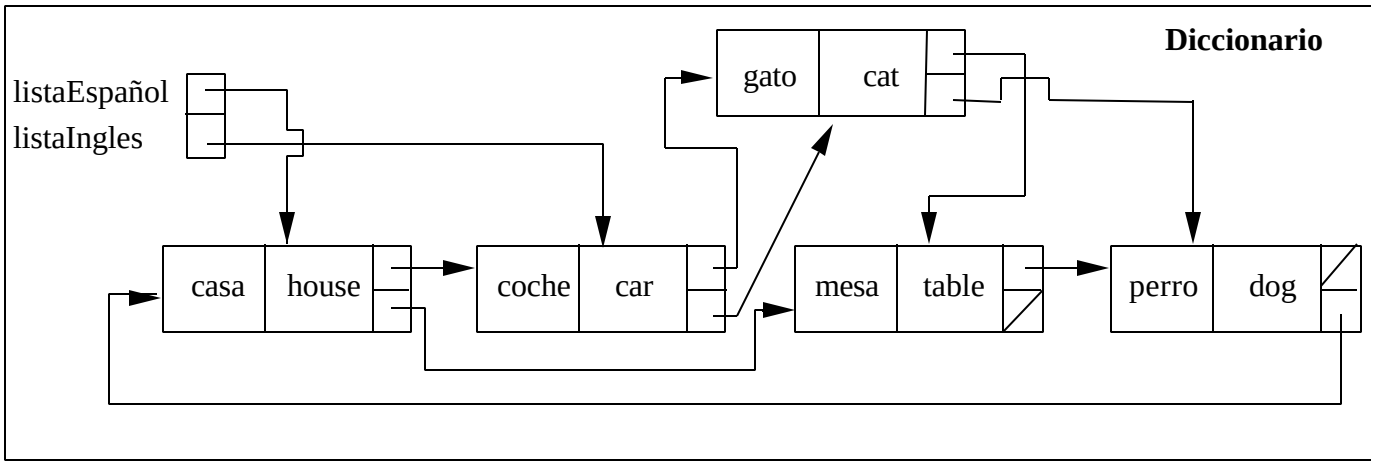
Cada palabra en castellano, (y respectivamente, cada palabra en inglés) se encuentra en el diccionario una sola vez, por tanto, una misma palabra no puede tener dos traducciones distintas

En el siguiente ejemplo se puede observar cómo la lista española contiene las palabras *casa coche, mesa* y *perro* (por orden alfabético en lengua española) y la lista inglesa contiene las palabras *car*, *dog*, *house* y por último a *table* (por orden alfabético en lengua inglesa).



Se pide :
a) La(s) clase(s) necesarias para recoger la especificación definida.
b) El diseño y la implementación en Java del método *añadirPalabra* de la clase Diccionario. Este método recibirá como entrada, una palabra en castellano y la correspondiente palabra en inglés. Si las palabras no están en el diccionario (ninguna de las dos) las inserta conservando el orden alfabético. Pero si existiese ya alguna de las dos palabras, ya sea la de castellano o la de inglés, produce un error.

En la siguiente figura se puede ver cómo quedaría el diccionario anterior tras aplicar el método insertar sobre el diccionario anterior con las palabras *gato/cat.*



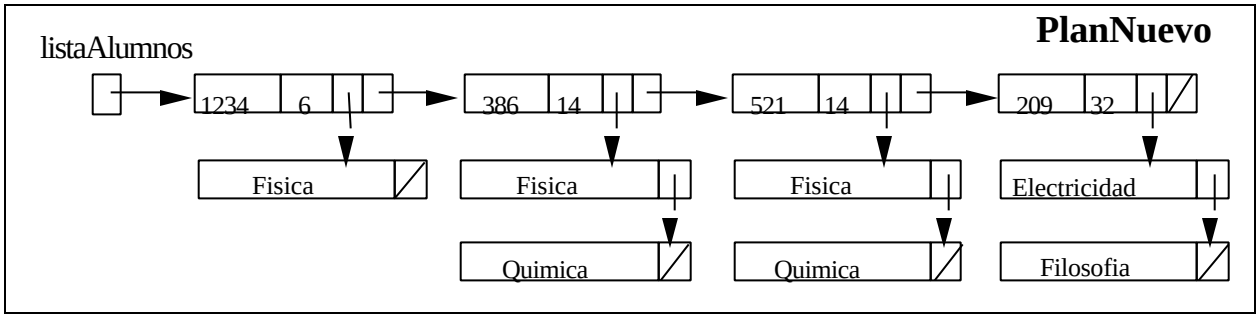
La cabecera del método a desarrollar es la siguiente:

```
public void anadirPalabra(String p1, String p2)
```

(J96) Disponemos en clase denominada PlanNuevo que representa una lista que contiene información referente a alumnos del Plan Nuevo.

Cada alumno está representado mediante el *número de expediente*, el *número de créditos* que ha obtenido hasta el momento, y una *lista de asignaturas* que ha superado. Los alumnos **están ordenados** ascendentemente según el *número de créditos* conseguidos, y aquellos que han obtenido el mismo número de créditos por orden creciente de *número de expediente*.

Ejemplo:



Se pide el diseño e implementación en Java de:

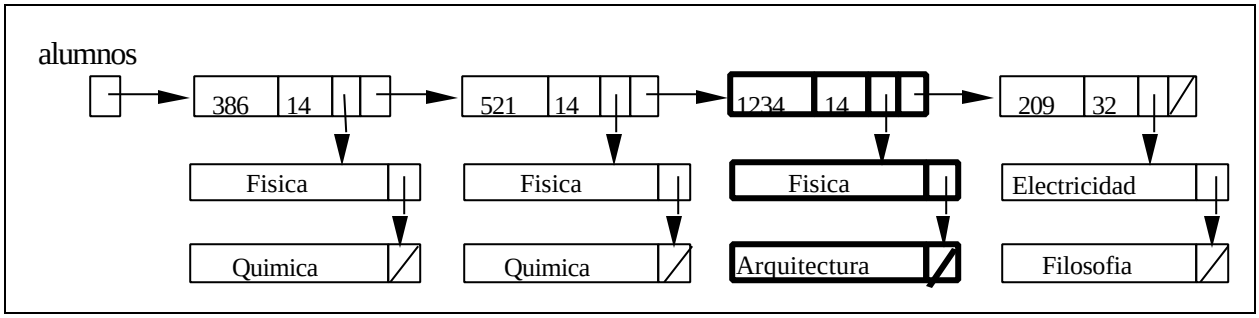
a) La(s) clase(s) necesarias para recoger la especificación definida.

b)* El método *superarAsignatura correspondiente a la clase PlanNuevo

```
 public  void superarAsignatura(int pnExp, String
ptit, int pcred)
```

- El método recibe como entradas un *número de expediente*, una *asignatura* y el *número de créditos* que corresponden a esa asignatura.
- La actualización de la lista de alumnos se debe hacer preservando el orden establecido (objetos Alumno ordenados por nº de créditos y a créditos iguales ordenados por nº de expediente).
- Si el *número de expediente* del alumno no existe previamente en la lista se debe de crear un nuevo objeto para ese alumno e insertarlo.
- Si el alumno existe y va a rebasarse el *número total de créditos* (300) se produce un error.
- Se produce un error si se pretende superar una asignatura que ya había sido *superada previamente* por el alumno.

En la siguiente figura se puede ver cómo quedaría la lista de alumnos anterior tras ejecutar **`superarAsignatura (1234,"Arquitectura", 8)`**



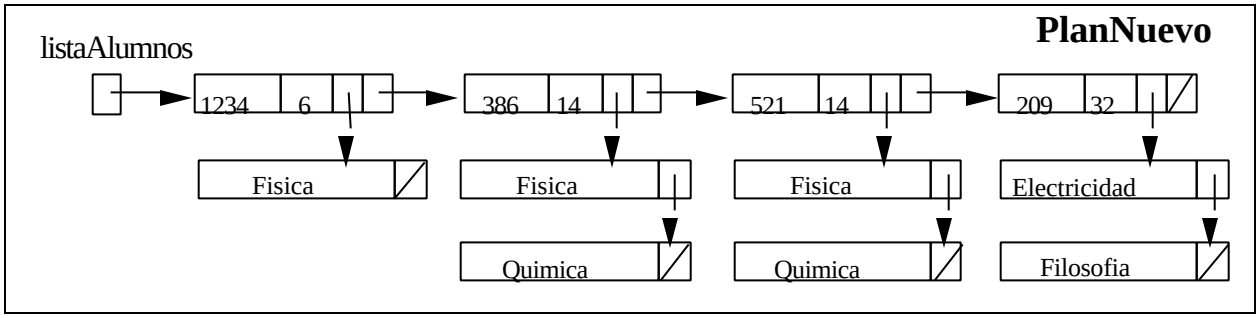
Vamos a suponer que tenemos implementado en método situarAlumno en la clase PlanNuevo.

```
public class PlanNuevo extends Object {
public void situarAlumno(Alumno pAl)
-- Pre    “pAl” apunta a un objeto cuyo campo
sigAlumno vale null.
-- Pos: el objeto apuntado por “pAl” se ha situado
en la
--“lista” de la clase PlanNuevo de acuerdo al orden
creciente por el
-- campo credits y a creditos iguales por orden
creciente del campo nExp
```

(S96) Disponemos en clase denominada PlanNuevo que representa una lista que contiene información referente a alumnos del Plan Nuevo.

Cada alumno está representado mediante el *número de expediente*, el *número de créditos* que ha obtenido hasta el momento, y una *lista de asignaturas* que ha superado. Los alumnos **están ordenados** ascendentemente según el *número de créditos* conseguidos, y aquellos que han obtenido el mismo número de créditos por orden creciente de *número de expediente*.

Ejemplo:



Se pide el diseño e implementación en Java de:

a) La(s) clase(s) necesarias para recoger la especificación definida.

b) El **diseño e implementación** en Java del **método** ***superarAsignatura correspondiente a la clase PlanNuevo***

```
public void superarAsignatura (Alumno alNuevo,
Alumno alAntiguo)
```

Especificación:

ENTRADA:

- *alNuevo*: objeto alumno que se quiere insertar
  (el objeto apuntado tiene el valor **null** en el campo sigAlumno)
- *alAntiguo*: objeto de la lista *perteneciente al PlanNuevo* que tiene el mismo nº de expediente que el apuntado por *nuevo*.
  Si no hay ningún alumno en la lista con ese nº de expediente el valor de *alAntiguo* es **null**.

SALIDA:

Si el objeto *alAntiguo* tiene valor **null**, entonces el objeto *alNuevo* ha sido colocado en la lista *perteneciente al PlanNuevo* según el siguiente orden:

ascendentemente según el **número de créditos** conseguidos, y para aquellos que han obtenido el mismo número de créditos por orden creciente de **número de expediente.**

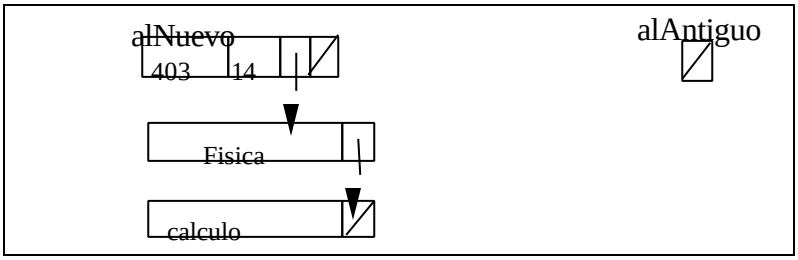
Si por el contrario el objeto *alAntiguo* es distinto de **null**, es porque el objeto estaba previamente en la lista. Entonces se deben considerar dos casos:

- La información que se tenía en la lista y la que se quiere colocar coinciden (las asignaturas superadas deben estar en el mismo orden):

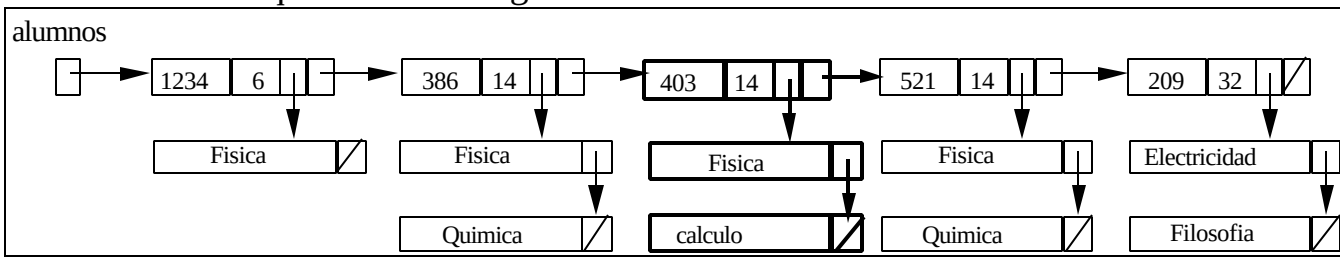
  no se ha hecho nada.
- No coinciden la información que se tenía y la que se pretende incluir:

  se ha generado un error.

Por ejemplo, si se colocara el siguiente objeto *alNuevo*, con *alAntiguo*=null, en la lista:

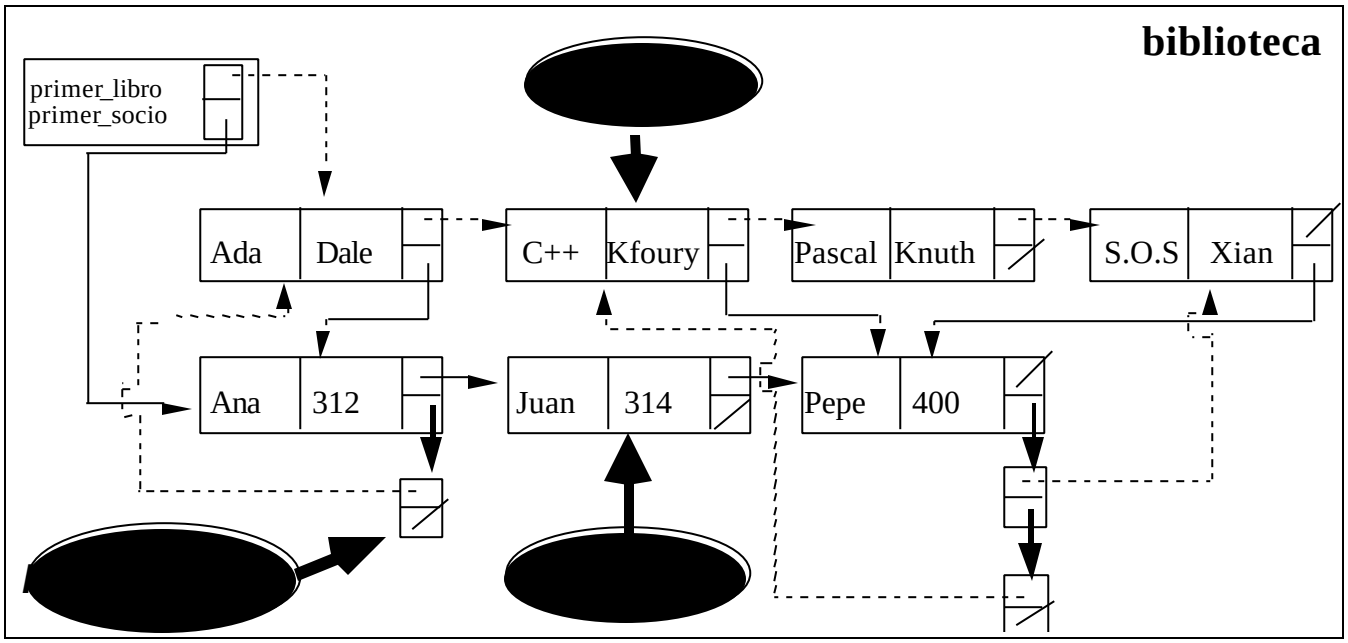


La lista *Alumnos* quedaría de la siguiente forma:



(J97)) Disponemos de una clase que representa una biblioteca. Dicha clase consta de la lista de libros que hay en la biblioteca, y de la lista de socios que pertenecen a ella. Cada una de estas listas está ***ordenada***, la de socios por el ***número de carnet*** y la de libros por ***orden alfabético del título del libro***.

La figura siguiente muestra una biblioteca que dispone de 4 libros y tiene 3 socios:



Cada elemento de la **lista de libros** consta de título, autor y una variable que da acceso al socio que lo tiene en su poder, (si el libro no está prestado esa variable es null). Cada elemento de la **lista de socios** consta de nombre, número de carnet y la **lista de préstamos** del socio (libros que ha sacado de la biblioteca). Cada elemento de una lista de prestamos consta de una variable al libro en cuestión (al elemento de la lista de libros).

Se pide :

a) La(s) clase(s) necesarias para recoger la especificación definida.

b) El **diseño e implementación** en Java del método **borrarSocio** de la clase Biblioteca, tal que dado el carnet de un socio realiza la siguiente operación:

- Si el socio no está en la lista de socios, se genera un error (ese socio no existe).

- Si el socio está, primero devuelve los libros que tuviera prestados (los pone como NO prestados) y segundo, borra el socio de la lista de socios.

```
public void borrarSocio(Socio pSoc)
```

(S97) Disponemos de una clases ***Lista*Ligada** que implementa una lista ligada y **ListaLigadaItr** que implementa los métodos para iterar sobre una lista ligada, tal y como se especifica en la primera página de este recopilatorio de ejercicios.

Se pide **Diseñar e implementar** en Java el método **`eliminaRepetidos`**, tal que dado un elemento de tipo entero, realice la siguiente operación:

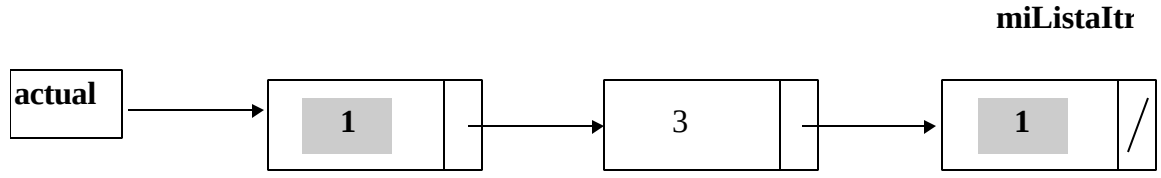
- Si en la lista no hay ningún nodo con ese elemento, genera un error.
- Si en la lista hay varios nodos con ese elemento, elimina todos ellos salvo la primera aparición.
- Si hay exactamente un nodo con ese elemento no hace nada.

```
public void eliminarRepetidos(int pInt)
```

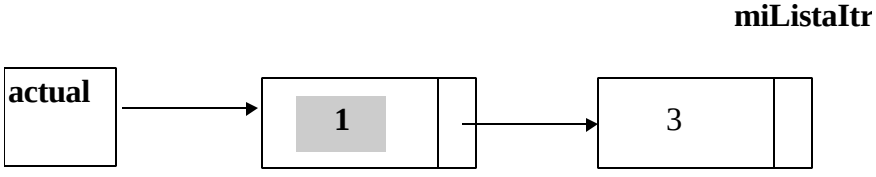
Ejemplo:

Dado un objeto iterador miListaItr de la clase ListaLigadaItr.

- `miListaItr.eliminarRepetidos(1)`

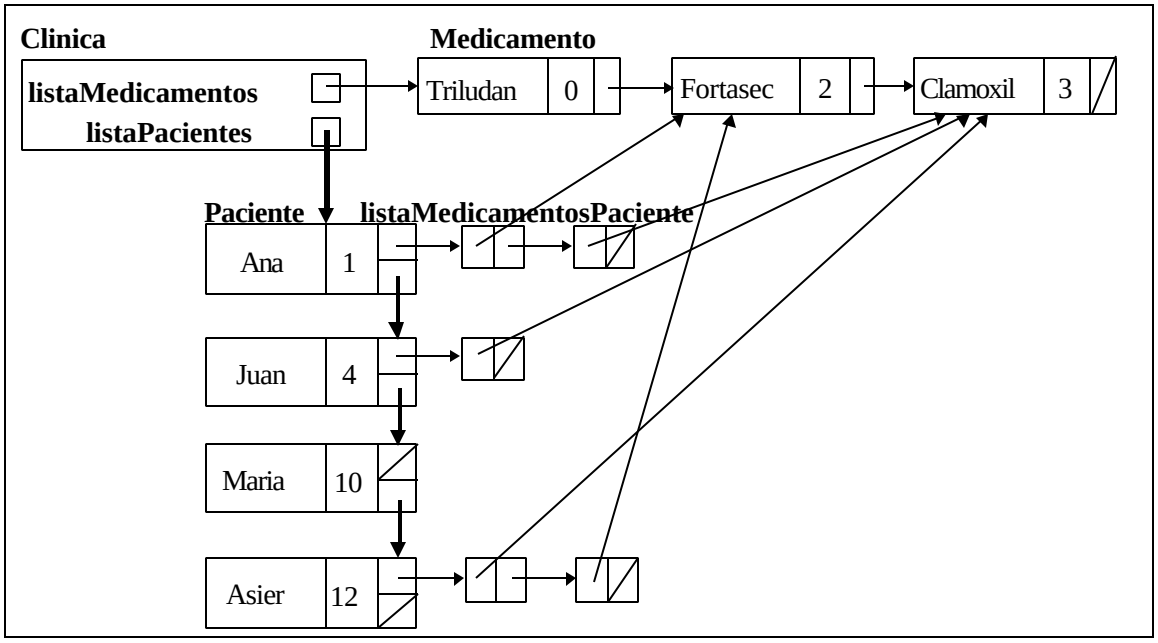


tras la ejecución del método `miListaItr` la lista quedaría de la siguiente forma:



(J98) Disponemos de una clase *Clínica* que representa una clínica. Dicha clase consta de la lista de pacientes, y de la lista de medicamentos. La lista de pacientes está ***ordenada*** por el ***número de cama que ocupa el paciente***.

La figura siguiente muestra una clínica que tiene 4 pacientes y 3 medicamentos:



Cada elemento de la **lista de pacientes** consta de nombre, número de cama y una **lista de sus medicinas** (si el paciente no toma medicinas esa lista es null). Cada elemento de la **lista de medicamentos** consta del nombre de la medicina y del número de pacientes que la toman.

Para un paciente concreto, cada elemento de la **lista de sus medicinas** consta de una referencia a un objeto medicamento (de la lista de medicamentos).

Se pide :

a) La(s) clase(s) necesarias para recoger la especificación definida.

b) El **diseño e implementación** en Java del procedimiento **añadirMedicina** perteneciente a la clase Clínica. Este método recibe como parámetros un objeto paciente que existe en la clínica y un objeto medicamento:

- Si en la lista de medicamentos no se encuentra el nuevo medicamento, entonces se añade a la lista con valor 1 en el campo nPacientes.
- Si en la lista de medicamentos ya se encontraba el nuevo medicamento, entonces aumenta en una unidad el campo nPacientes.

- Añade a la lista de las medicinas del paciente, el nuevo medicamento.

public void añadirMedicina(Paciente pPac, String pMedicina)

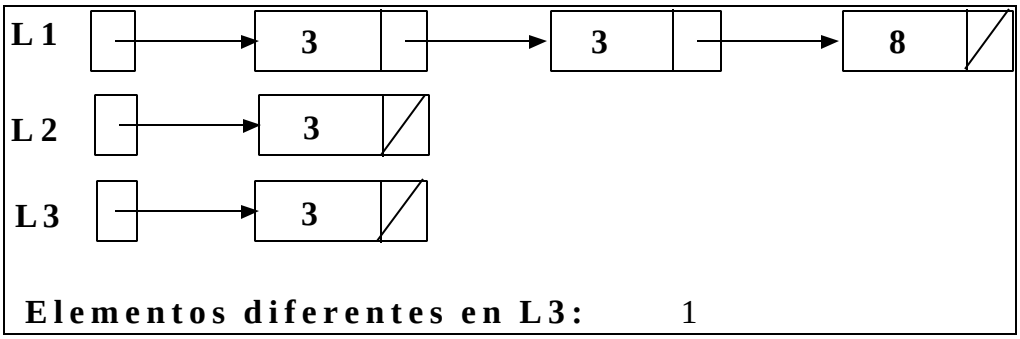
--Pre: *paciente* es un objeto de la lista de pacientes de la clínica

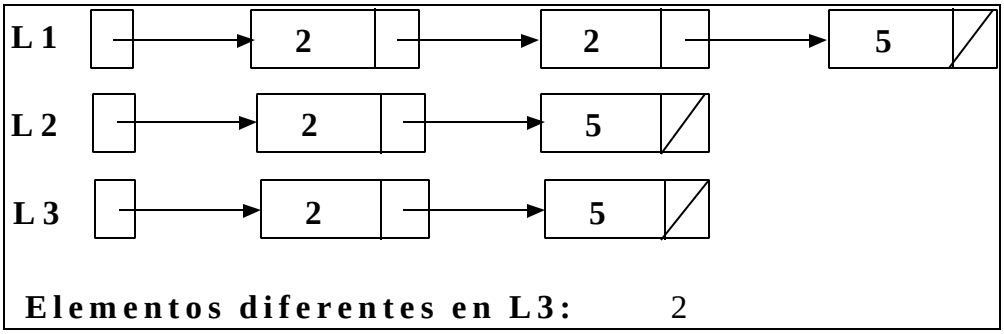
(S98) Disponemos de una clases ***Lista*Ligada** que implementa una lista ligada y **ListaLigadaItr** que implementa los métodos para iterar sobre una lista ligada, tal y como se especifica en la primera página de este recopilatorio de ejercicios. Los elementos de la lista ligada están *ordenadas de menor a mayor***.**

Diseñar y escribir en Java un método tal que dadas dos instancias de la clase ListaLigadaItr, devuelva otro objeto ListaLigadaItr que contenga **la intersección** de las dos anteriores y el **número de elementos diferentes que tiene la lista devuelta**.

Habrá que tener en cuenta que en una misma lista ordenada los elementos pueden aparecer repetidos dos o más veces.

Ejemplos **(La lista** L3 **es la intersección de las listas** L1 **y** L2**):**





(1) Sobre la misma clase introducida en el ejercicio **J94** (y en **S95**), **diseña e implementa** un método en la clase Diccionario tal que dada una palabra en español y otra en inglés:

- Si las dos palabras se encuentran en el mismo nodo, las elimine de la estructura.
- Si están en nodos diferentes genere un error.
- Si alguna de las palabras no se encuentra en el diccionario, genere un error.

(2) Considerando ahora las clases utilizadas en los ejercicios **J95** (y **J97**), **diseña e implementa** un método de la clase Biblioteca tal que dada la información de un libro:

- Realice la devolución del libro a biblioteca y devuelva en un parámetro de salida el número de socio que lo tenía en préstamo.
- Genere un error si el libro indicado no se encuentra en la biblioteca.
- Genere un error si el libro no se encuentra en préstamo (considerar que no puede haber dos libros en la biblioteca con la misma información).

(3) Teniendo en cuenta las clases definidas en el ejercicio **S94**, **diseñar e implementar** un método tal que dada una asignatura, una nota y el nombre y apellido de un alumno:

- Introduzca la asignatura para ese alumno. En este caso vamos a considerar que las asignaturas se encuentran ordenadas alfabéticamente para cada alumno.
- Genere un mensaje de error si el alumno no existe.
- Modifique la nota si la asignatura estaba en las asignaturas del alumno pero con suspenso.
- Genere un mensaje de error si la asignatura ya estaba aprobada.

(4) Sobre las mismas clases del ejercicio anterior **diseña e implementa** un método tal que dada una asignatura, y el nombre y apellido de un alumno:

- Elimine la asignatura de ese alumno. También consideraremos que las asignaturas se encuentran ordenadas alfabéticamente para cada alumno.
- Genere un mensaje de error si el alumno no existe.
- Genere un mensaje de error si la asignatura no está las asignaturas del alumno.

(5) La clase definida en **S97** contiene un conjunto de números desordenados. **Diseña e implementa** un método que ordene los elementos de una de estas listas de números *de menor a mayor*.

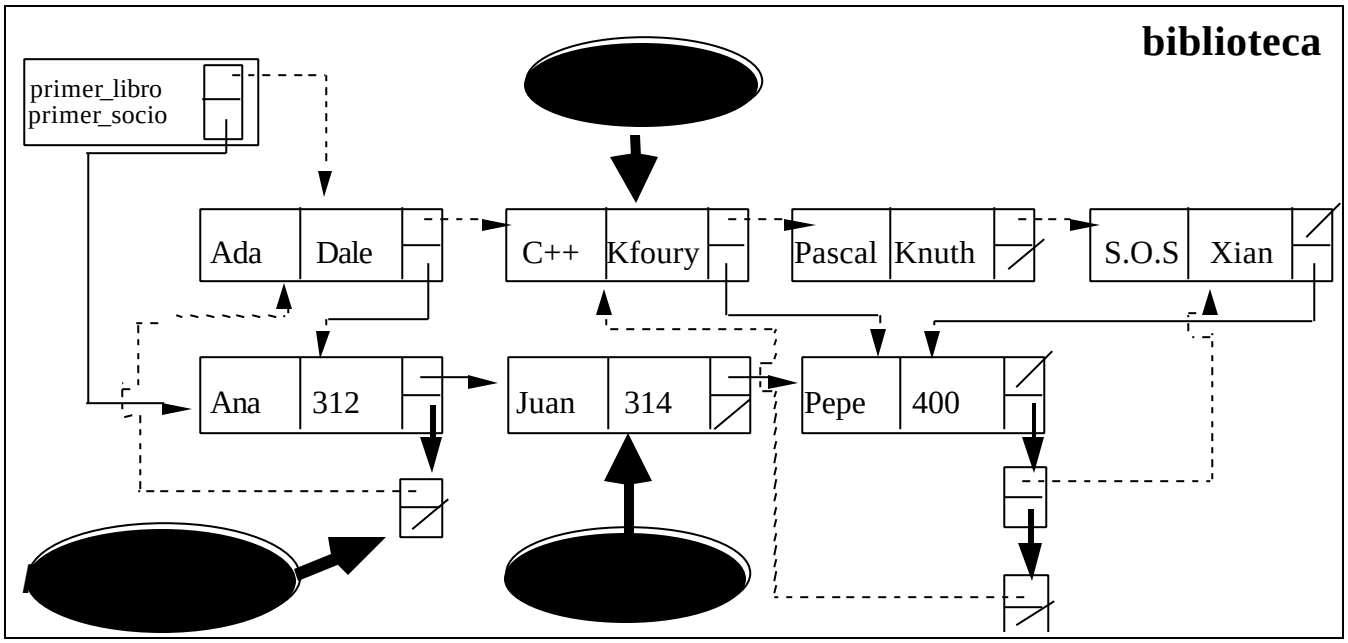
(6) Dadas dos instancias de la clase ListaLigadaItr(con parámetro ListaLigada) con la misma estructura del ejercicio anterior, **diseña e implementa** un método que obtenga una tercera lista (de idéntica organización) que contenga los mismos números presentes en las listas de entrada.

(7) Considera ahora las clases que se muestran en **J98**. Se trata de **diseñar e implementar** un método en la clase Clínica, tal que dada una medicina, la elimine de la clínica. Si la medicina no existe, generar el correspondiente error.

(8) Dadas dos instancias de la clase ListaLigadaItr(con parámetro ListaLigada) de números ordenados ascendentemente, como las del ejercicio **S98**, obtener la unión de las mismas sin que la lista unión tenga elementos repetidos y conservando el orden ascendente.

(9)) Disponemos de una clase que representa una biblioteca. Dicha clase consta de la lista de libros que hay en la biblioteca, y de la lista de socios que pertenecen a ella. Cada una de estas listas está ***ordenada***, la de socios por el ***número de carnet*** y la de libros por ***orden alfabético del título del libro***.

La figura siguiente muestra una biblioteca que dispone de 4 libros y tiene 3 socios:



Cada elemento de la **lista de libros** consta de título, autor y una variable que da acceso al socio que lo tiene en su poder, (si el libro no está prestado esa variable es null). Cada elemento de la **lista de socios** consta de nombre, número de carnet y la **lista de préstamos** del socio (libros que ha sacado de la biblioteca). Cada elemento de una lista de prestamos consta de una variable al libro en cuestión (al elemento de la lista de libros).

Se pide :

a) La(s) clase(s) necesarias para recoger la especificación definida.

b) El **diseño e implementación** en Ada del procedimiento **cambiarPrestatario** de la clase Biblioteca, tal que dado el título de un libro y el carnet de un socio atienda la petición de cambio de prestatario de la siguiente forma:

- El libro estaba prestado a un socio y se ha realizado una doble acción:
    1. El libro ya no figura como prestado al socio que lo tenia.
    2. El libro figura como prestado al socio cuyo nº de carnet nos facilitan.

- Si el libro estaba prestado justamente al socio cuyo carnet nos dan, la biblioteca no se modifica.
- Si el libro no estuviera en la biblioteca, se generará el error *libro_no_existe* (la biblioteca puede no tener libros).
- Si el libro no estuviera prestado, se generará el error *libro_no_prestado.*
- Si el carnet no correspondiera al de ningún socio se generará el error *socio_no_existe*.

```
public void cambiarPrestatario (String pLibro,
                                int pSocio)
```