Apellidos:	***Nombre:***
Grupo:	***DNI:***
Duración: 2:30 h.	

Ejercicio 1	Ejercicio 2	Ejercicio 3	TOTAL

Ejercicio 1 [3 puntos].

1.1: **[1 punto]** Dada una tabla hash donde las colisiones se resuelven por Hashing doble y cuyas funciones de dispersión primaria y secundaria son respectivamente:

$$h(k) = k \bmod 13$$
$$d(k) = 7 - k \bmod 7$$

Determinar cómo quedaría dicha tabla después de la siguiente secuencia de inserciones.

Seq.	Nombre	Apellido	DNI
1	Mariano	Rajoy	41.002.118
2	Alfredo	P. Rubalcaba	8.424.241
3	Cayo	Lara	85.669.022
4	Josep A.	Durán	35.875.644
5	Rosa	Díez	41.002.159
6	Esperanza	Aguirre	1.284.332
7	Tomás	Gómez	22.467.231
8	Felipe	González	33.349.173

1.2: **[1 punto]** Dada la información contenida en la siguiente tabla, construir un árbol B de grado 5 (considerado como clave el sueldo) y representarlo con formato de fichero.

	Nombre	Apellido	Sueldo (M $)
0	Abraham	Duarte	1
1	Raúl	Cabido	2
2	José	Vélez	3
3	Ángel	Sánchez	4
4	Hulk	Hoogan	5
5	Santiago	Segura	6
6	Eric	Roberts	7
7	Novak	Djokovic	8
8	Amanda	Ooms	9
9	Nan	Yu	10
10	Randy	Couture	11
11	Scott	Adkins	14
12	Terry	Crews	15
13	Liam	Hemsworth	16
14	Mickey	Rourke	17
15	Jason	Statham	18
16	Jet	Li	19
17	Dolph	Lundgren	20
18	Chuck	Norris	22
19	Jean-Claude	Van Damme	26
20	Bruce	Willis	27
21	Sylvester	Stallone	28
22	Arnold	Schwarzenegger	32

NOTA: Para que el ejercicio sea evaluado correctamente, es necesario representar todos los pasos intermedios.

1.3: **[1 punto]** Una agencia de viajes quiere almacenar en una estructura de datos eficiente la información de vuelos con origen en Madrid y destino en algunas de las ciudades más importantes del mundo. En concreto, quiere determinar qué ciudades están a una distancia (en tiempo) menor que un umbral dado. Determine la estructura de datos más adecuada para resolver este problema y represente paso a paso cómo se iría conformado dicha estructura si se asume que la información se introducen en el orden dado por la tabla.

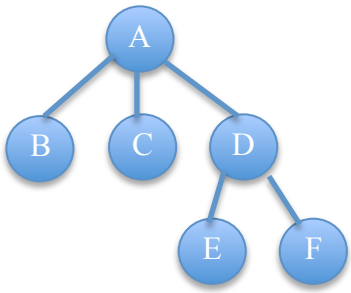
Nº Horas	Destino	Tipo Avión	Nº Pasajeros
2	Barcelona	Boening 727	125
5	Munich	Boeing727	125
10	Tokio	Airbus 340	300
13	Singapure	Airbus 340	300
1	Valencia	Boeing 727	125
12	Pelin	Airbus 340	300
16	Melbourne	Airbus 340	300
14	Tasmania	Airbus 340	300
15	Honolulu	Airbus 340	300

NOTA: Para que el ejercicio sea evaluado correctamente, es necesario representar todos los pasos intermedios.

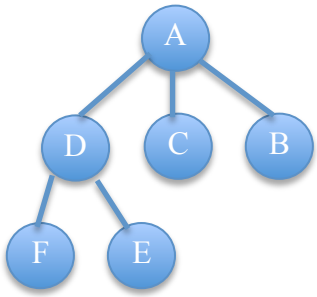
Ejercicio 2 [3 puntos].

Aumentar la funcionalidad de los árboles n-arios (`LinkedTree`) añadiendo los siguientes métodos:

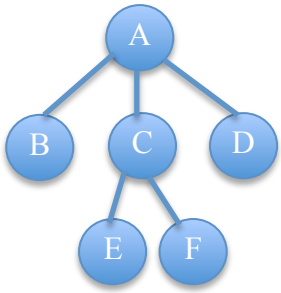
a) **[1.5 puntos]** Método que determine si dos árboles son iguales. Para ello, debe comprobar por niveles si los elementos son iguales (independientemente del orden). Por ejemplo, dado el árbol T1, el método implementado debería determinar que es igual a T2 y distinto a T3.



Árbol T1

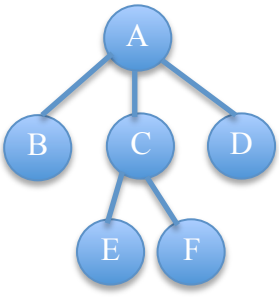


Árbol T2

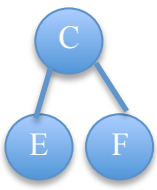


Árbol T3

b) **[1.5 puntos]** Método que reciba el `Position` de un nodo y devuelva (en una copia) el subárbol enraizado en dicho nodo. Por ejemplo, dado el árbol T1 de la figura, si se recibe como argumento de entrada el position del nodo `C`, devolvería el subárbol T2.



Árbol T1



Subárbol T2

Ejercicio 3 [4 puntos].

Implementar una clase en Java (`GestorViajes`) que permita gestionar en memoria una serie de viajes de una agencia, donde cada viaje se representa con la ciudad origen, destino y la duración del viaje (ver clase `Viaje`). La información se deberá almacenar de tal forma que se pueda consultar eficientemente todos los viajes que tienen una determinada ciudad origen o todos los que tienen una determinada ciudad destino.

Se pide implementar:

a) **[0.5 puntos]** Definir el tipo (clase) `GestorViajes`, justificando la elección. También se deben definir todos los tipos de datos auxiliares que se consideren necesarios.

b) **[1.0 punto]** Implementar las operaciones `crearGestorViajes` (puede ser el constructor de la clase) e `insertarViaje`, indicando la complejidad de cada una de ellas. Se deberán implementar también todas las operaciones auxiliares necesarias.

c) **[1.5 puntos]** Implementar las operaciones `getDestinos(String ciudad)` y `getOrigenes(String ciudad)`, que devuelven respectivamente todos los viajes que tienen una determinada ciudad origen y todos los viajes que tienen una determinada ciudad destino.

d) **[0.5 puntos]** Implementar una operación que devuelva de la forma que considere más adecuada todos los viajes almacenados.

e) **[0.5 puntos]** Implementar una operación que devuelva de la forma que considere más adecuada las ciudades en las que hay, al menos, un viaje.

NOTA: Es imprescindible diseñar el código primiando el tiempo de ejecución de las consultas, aunque esto requiera del uso varias estructuras de datos

```
class Viaje {
  private String origen;
  private String destino;
  private int duracion;

  public Viaje(String o, String d, int dur){
    origen = o;
    destino = d;
    duracion = dur;
  }

  public String getOrigen(){
    return origen;
  }

  public String getDestino(){
    return destino;
  }

  public String getDuracion(){
    return duracion;
  }
}
```

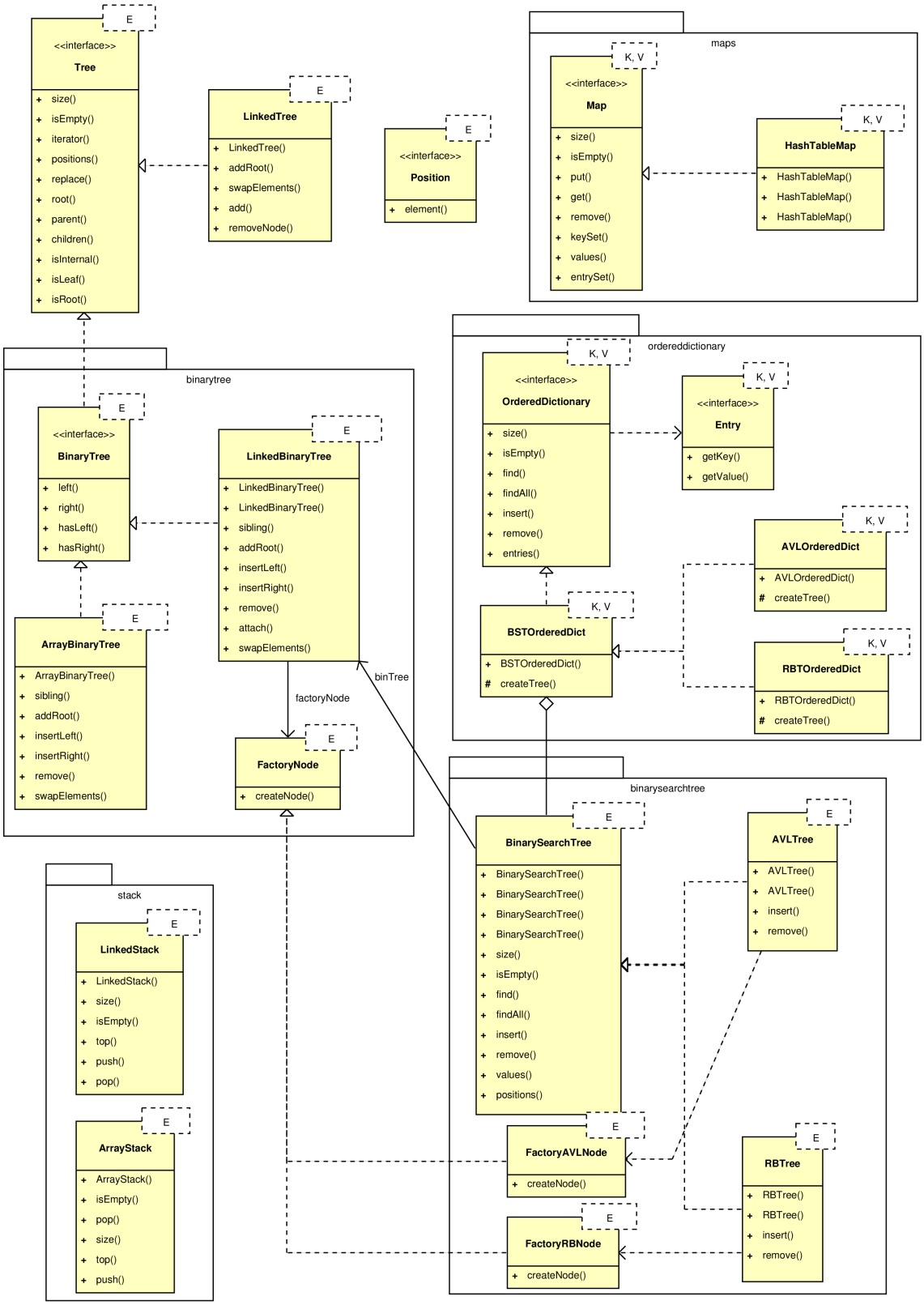


Diagrama de Clases correspondiente a las estructuras presentadas en clase

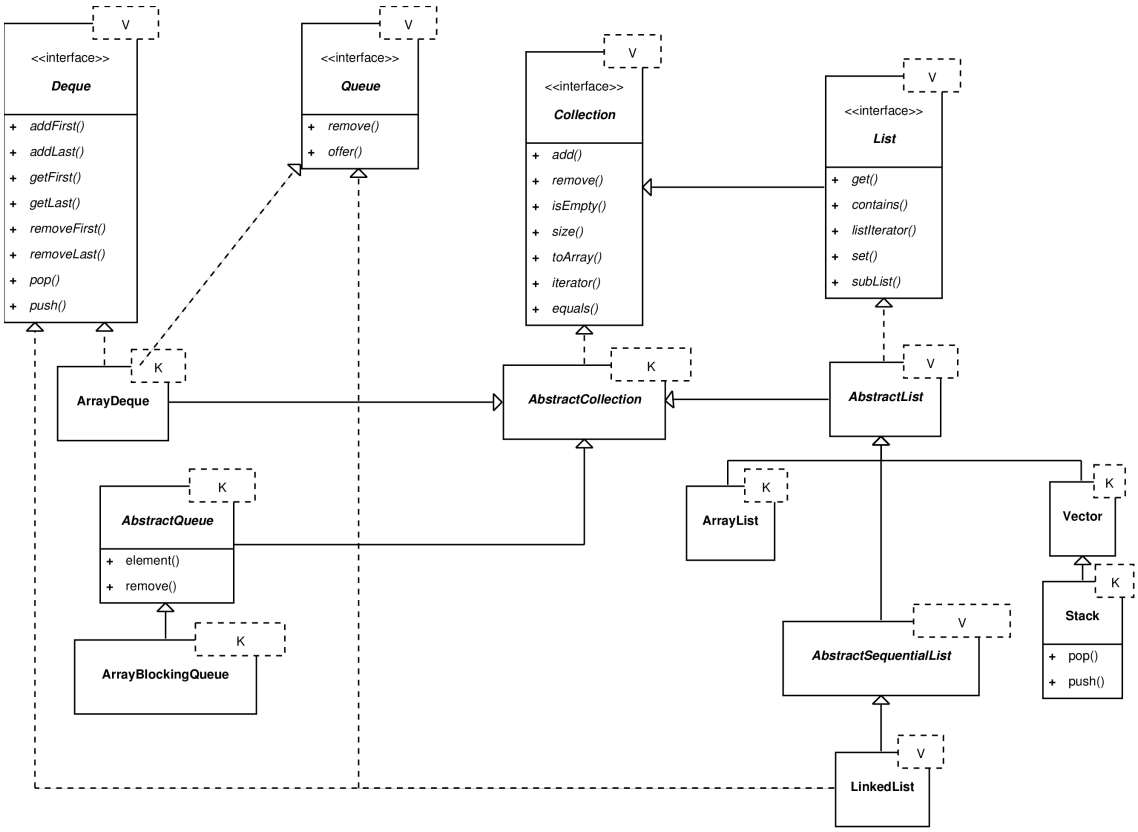


Diagrama de clases correspondiente a las estructuras de datos lineales de Java