Ejercicio 1 [3.0 puntos]. Eva López Puente

1.1: **[1.5 puntos]** Dado el fichero que aparece al final del enunciado, diseñar los índices que considere adecuados para poder hacer búsquedas de complejidad logarítmica en dicho fichero, tanto por número de horas, como por destino. Se pide también la representación de dichos índices en formato fichero.

	Nº Horas	Destino	Tipo Avión	Nº Pasajeros
0	8	Barcelona	Boening 727	125
1	3	Munich	Boeing727	125
2	6	Tokio	Airbus 340	300
3	21	Singapure	Airbus 340	300
4	15	Valencia	Boeing 727	125
5	17	Pekin	Airbus 340	300
6	16	Melbourne	Airbus 340	300
7	44	Tasmania	Airbus 340	300

1.2: **[1.5 puntos]** Se quieren introducir los datos que aparecen al final del enunciado en un tabla hash de tamaño 10, donde la *clave* es el DNI y el *valor* el registro corresponidente. Se supone que el DNI es un `String`, por lo que es necesario redefinir el método `hashCode()`, NO CÓDIGO por ejemplo, basado en suma de componentes de 3 en 3, quitando la letra del DNI. Como función de compresión se debe usar $h(k) = k \bmod Tam\_Tabla$ y las colisiones se tienen que resolver por prueba cuadrática. Determinar cómo quedaría dicha tabla después de seguir la secuencia de inserciones descrita en Seq.

Seq.	Nombre	Apellido	DNI
1	Mariano	Rajoy	41.002.118P
2	Pedro	Sánchez	8.424.241Q
3	Pablo	Iglesias	85.669.022R
4	Josep A.	Durán	35.875.644S
5	Rosa	Díez	41.002.159T
6	Alberto	Garzón	85.669.022R
7	Alfred	Bosch	22.467.231W
8	Uxue	Barcos	33.349.173X

NOTA: Para que los ejercicos sean evaluados con la máxima calificación, es necesario representar los pasos intermedios. Presentar el resultado final equivale a una calificación de cero en el ejercicio correspondiente.

Ejercicio 2 [3.5 puntos]

Se desea aumentar la funcionalidad de los árboles *n*-arios incorporando la posibilidad de hacer `clear` y `copy`.

a) **[1.0 puntos]** `clear`: que podrá invocarse sobre cualquier árbol *n*-ario (`LinkedTree` o `LCRSTree`), modificando para que quede vacío.

b) **[2.5 puntos]** `copy`: que recibe un árbol *n*-ario de cualquier tipo (`LinkedTree` o `LCRSTree`) y lo copia a cualquier otro formato. Por ejemplo, podría recibir un `LinkedTree` y podría devolver un `LinkedTree` o un `LCRSTree`

NOTA: Se valorará un buen diseño (de clases y métodos) para obtener la máxima calificación. Se recuerda que, por lo general, es un mal diseño tener métodos que compartan mucho código. Asimismo, es imprescindible que los métodos implementados sean lo más eficiente posible.

Ejercicio 3 [3.5 puntos]

La empresa WEB S.A. realiza portales web para terceros. Su solución actual organiza las páginas web de cada sitio en un árbol, donde cada nodo se corresponde con una página web que contiene un archivo HTML. Básicamente, cada página HTML contienen texto en castellano, imágenes y enlaces a otras páginas.

La empresa desea añadir un módulo de traducción que permitirá traducir los textos que aparecen en cada página HTML. Para ello, se desea que dicho modulo esté compuesto por una clase `Traductor` que permita insertar, modificar, buscar y eliminar traducciones de cada párrafo. Por ejemplo, un traductor a inglés, francés y alemán permitiría almacenar para la frase *"Pulse aquí para hacer acceder"* la traducciones *"Press here to login"*, *"Cliquez ici pour accéder"* y *"Klicken sie hier um auf"*. Así, cuando alguien solicite un nodo HTML, previamente se aplicará el traductor a cada frase que contenga, utilizando el objeto traductor que del idioma seleccionado por el usuario.

Se pide:

a) **[0.5 puntos]** Definir el tipo de datos (clases en Java) `Traductor`, de tal forma que la complejidad algorítmica de las operaciones requeridas sea la menor posible.

b) **[1.0 punto]** Implementar el método `añadir` que reciba un texto en castellano, su correspondiente traducción e idioma, y lo añada al `Traductor`. Por ejemplo

```
añadir(Hola, Hello, Inglés).
```

c) **[1.0 punto]** Implementar el método `traducir` que reciba un texto en castellano y el idioma destino y devuelva el texto traducido en dicho idioma. Por ejemplo

```
traducir(Pulse aquí, Inglés) -> Press here
```

d) **[1.0 punto]** Implementar el método `traducciones` que reciba un texto y devuelva todas las traducciones de dicho texto junto con el idioma.

```
traducciones("Pulse aquí") ->
[Press here,Inglés],[Cliquez ici,Francé],[Klicken hier,Alemán]
```

NOTA: Para crear la clase `Traductor` se podrá agregar en su interior cualquiera de las estructuras de datos estudiadas durante el curso, valorándose especialmente lo adecuado de la elección que se realice. No es necesario proporcionar el código de aquellas estructuras de datos desarrolladas durante el curso.

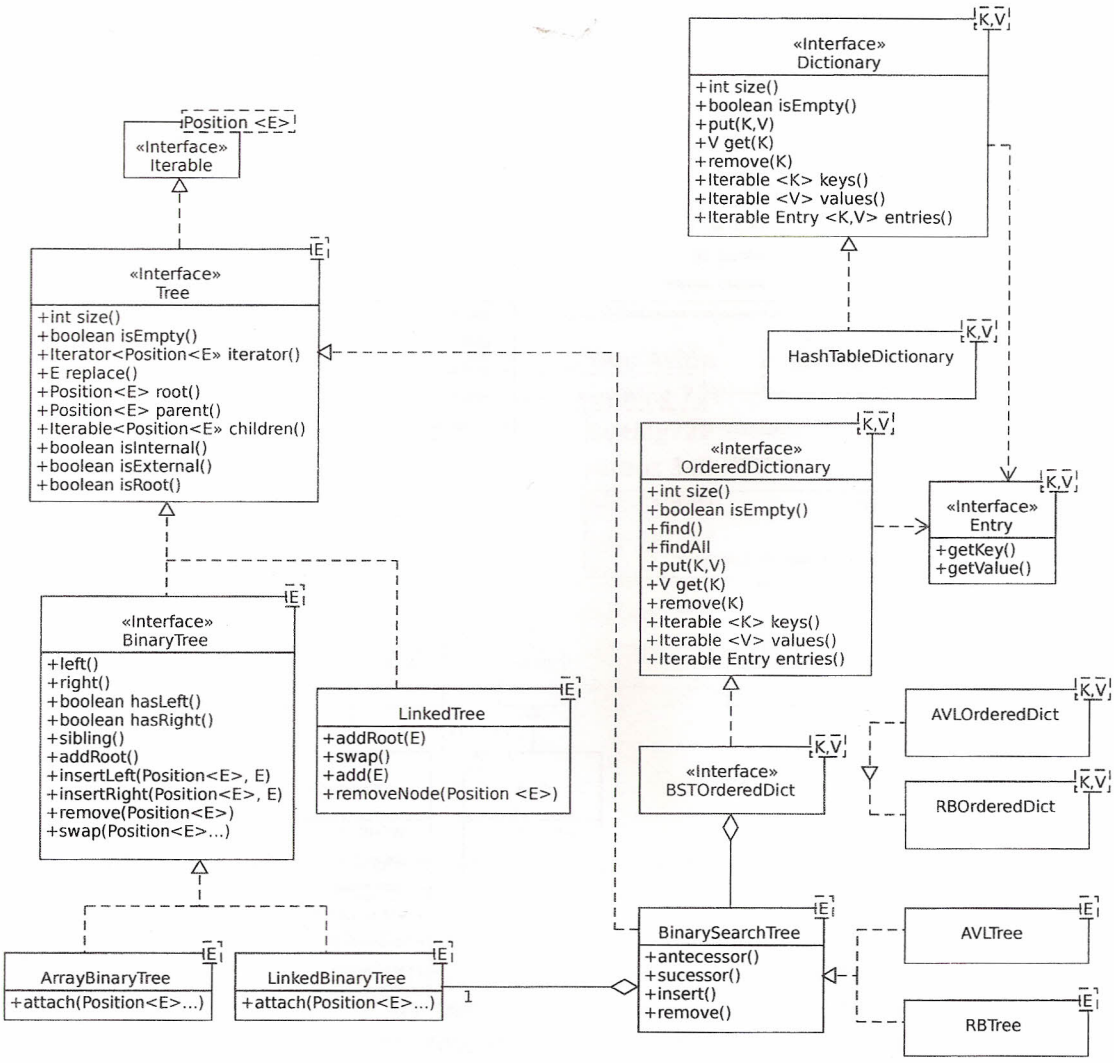

Diagrama de Clases correspondiente a las estructuras presentadas en clase

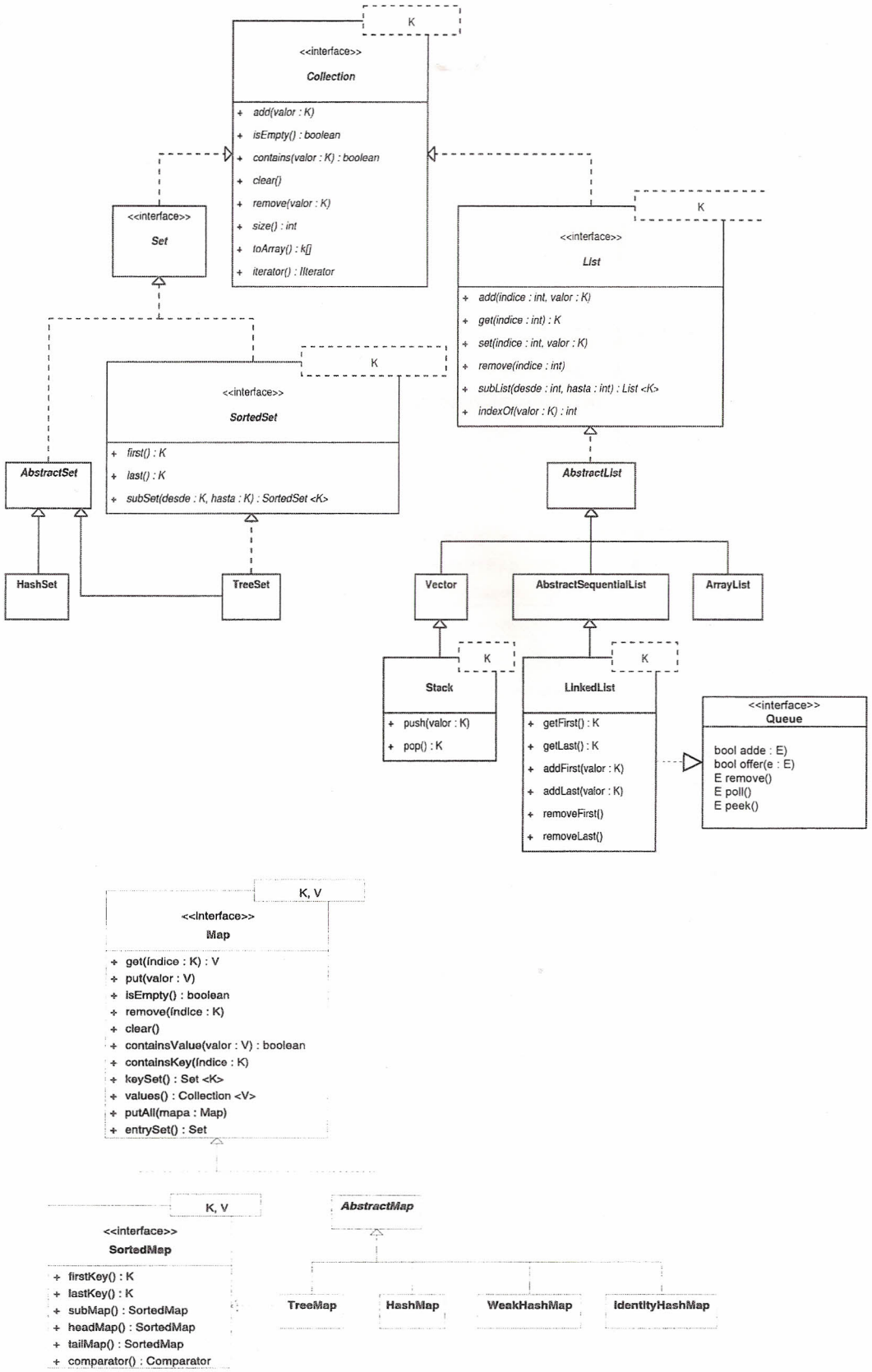

Diagrama de clases correspondiente a las estructuras propias de Java