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

Ejercicio 1	Ejercicio 2	Ejercicio 3	TOTAL

Ejercicio 1 [3 puntos].

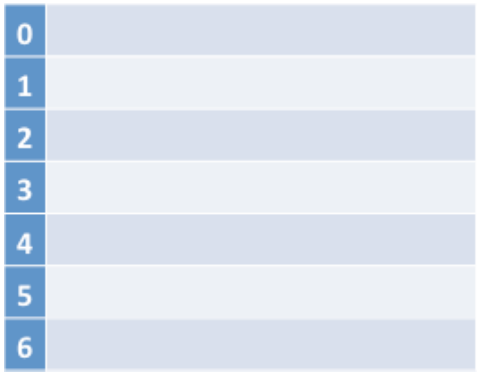
a) Dada una tabla hash de tamaño 7 cuya función de dispersión es h(x) = x mod tamaño. Determine cómo quedaría dicha tabla después de insertar la siguiente secuencia de claves:

{56, 92, 12, 7, 89, 71, 19, 23, 105, 60, 69, 27, 97}

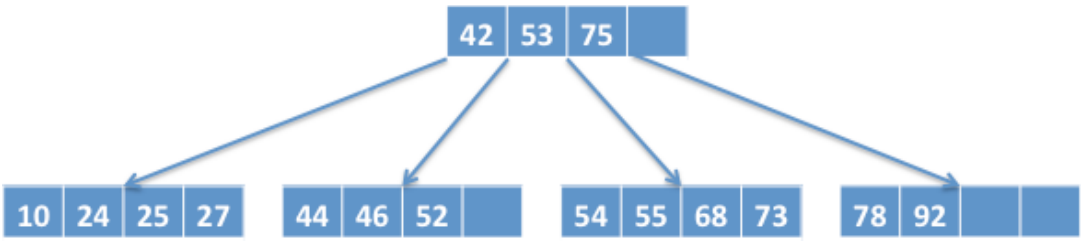
Considerando que las colisiones se resolveran mediante:

- Encadenamiento separado
- Prueba cuadrática

Se aplicará rehashing cuando el factor de carga sea 0.5. Indicar además con un '*' aquellas posiciones de la tabla donde ocurran las colisiones y explicar cómo se han tratado.



b) Dado el árbol B de orden 5, se pide insertar secuencialmente las siguientes claves: 13, 50, 76, 62 y 77, dibujando el árbol B resultante tras insertar cada una de las claves.



c) Insertar en un árbol binario de búsqueda rojo-negro la siguiente secuencia de claves

{10, 13, 18, 21, 9, 20, 24, 22, 23}

Para que el ejercico se considere válido, se debe mostrar el estado del árbol después de cada inserción, re-equilibrandolo en el caso de que fuera necesario

Ejercicio 2 [3 puntos].

Añadir a la clase `BinarySearchTree` el siguiente comportamiento:

```
/*Devuelve una posición al menor valor dentro del árbol*/
public Position<E> first()throws BoundaryViolationException{
            // TO-DO
}

/*Devuelve una posición al mayor valor dentro del árbol*/
public Position<E> last() throws BoundaryViolationException{

            // TO-DO
}

/*Devuelve una colección iterable con todos los valores
mayores que pos */
public Iterable<Position<E>> succesors(Position<E> pos)
                        throws InvalidPositionException{
            / TO-DO
}

/*Devuelve una colección iterable con todos los valores
menores que pos */
public Iterable<Position<E>> predecessors(Position<E> pos)
                        throws InvalidPositionException {
            // TO-DO
}
```

NOTA: Vista simplificada de las clases *BinarySearchTree* y *BTNode*, que pueden ser útiles para la resolución del ejercicio.

```
public class BinarySearchTree<E> {

	protected LinkedBinaryTree<E> binTree;
	protected Comparator<E> comparator; // comparator
	protected int size = 0; // number of entries

	public int size() {}
	public boolean isEmpty() {}

	public Position<E> find(E value) {}
	public Iterable<Position<E>> findAll(E value) {}

	public Position<E> insert(E value) {}
	public E remove(Position<E> pos){}

	public Iterable<E> values() {}
	public Iterable<Position<E>> positions() {}

	protected Position<E> treeSearch(E value, Position<E> pos){}

	protected BTNode<E> checkPosition(Position<E> p){}

}

public class BTNode<E> implements Position<E> {
	private E element;
	private BTNode<E> left, right, parent;

	public BTNode(E element, BTNode<E> parent,
			BTNode<E> left, BTNode<E> right) {}

	public E element() {}
	public void setElement(E o) {}

	public BTNode<E> getLeft() {}
	public void setLeft(BTNode<E> v) {}

	public BTNode<E> getRight() {}
	public void setRight(BTNode<E> v) {}

	public BTNode<E> getParent() {}
	public void setParent(BTNode<E> v) {}

}
```

Ejercicio 3 [4 puntos].

Se dispone de una lista de radares de tráfico para una gran ciudad que permiten obtener la información centralizada de vehículos infractores. Cada vez que tiene lugar una infracción, ésta se almacena como un objeto `Multa` para su posterior análisis:

```
public class Multa{
      private String Matricula;
      private float importeAPagar;
      private int idRadar;
      …
`
```

En concreto, el centro de control está interesado en poder llevar acabo las siguientes acciones:

- Recibir una sanción (multa) y almacenarla para su posterior análisis.
- Dado un conductor, identificado por su matrícula, consultar la cuantía a pagar por el mismo. La cuantía se calcula como la suma total de las multas de dicho conductor.
- Consultar en cualquier instante que radar es el que más multas ha puesto hasta el momento.

Para la resolución del ejercicio se pide:

a) Seleccionar la(s) estructura(s) de datos necesarias para resolver el problema.
b) Añadir los atributos necesarios para que la clase `CentroControl` pueda hacer uso de las estructuras de datos seleccionadas en el punto anterior
c) Implementar los métodos `setMulta()`, `getCuantia()` y `getRadarMasValioso()`.

```
public class CentroControl{
      public void setMulta(Multa m){}
      public float getCuantia(String Matricula){}
      public int getRadarMasValioso(String Matricula){}
}
```

NOTA: se tendrá en cuenta la complejidad algorítmica de la solución dada

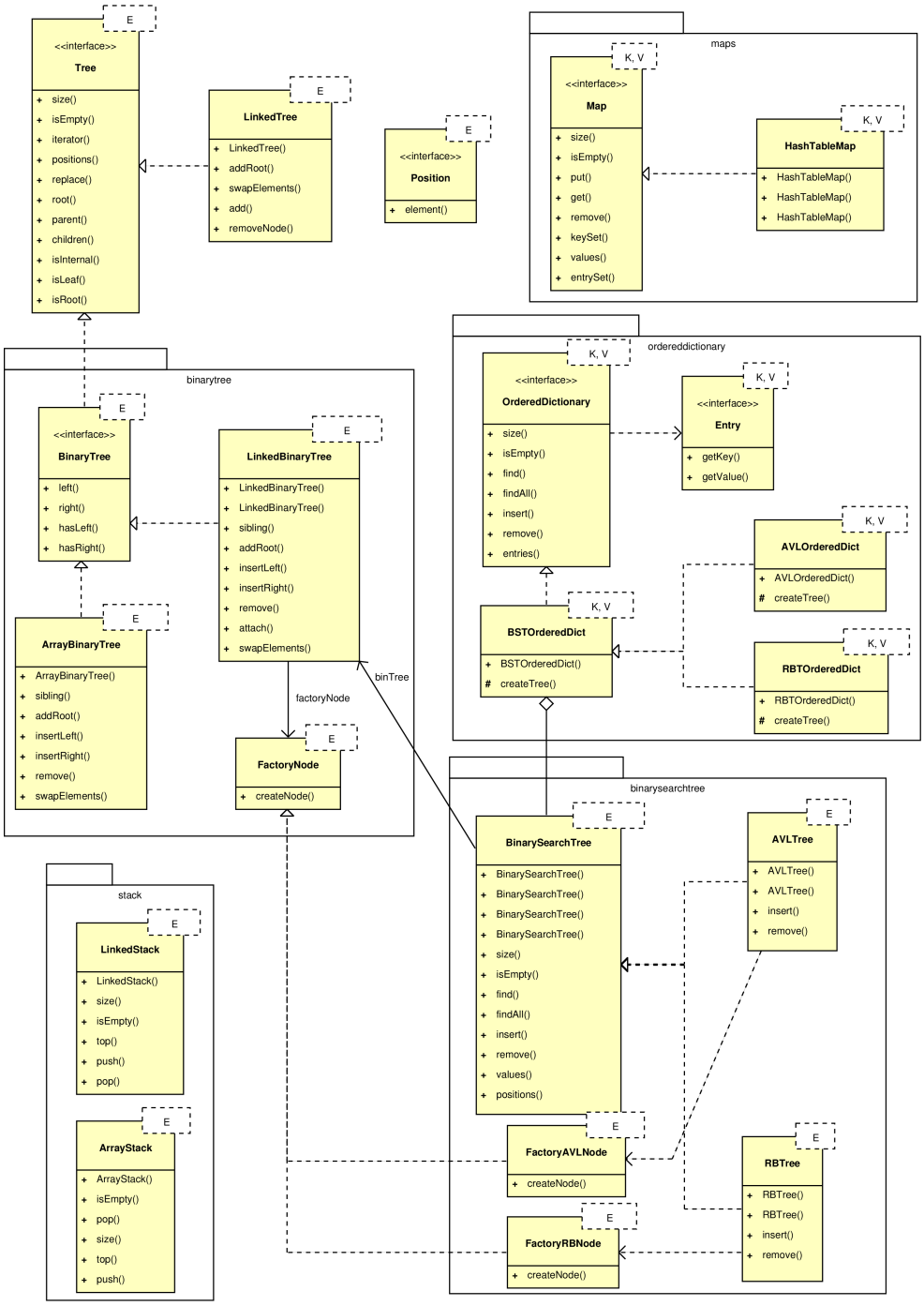


Diagrama de Clases correspondiente a las estructuras presentadas en clase

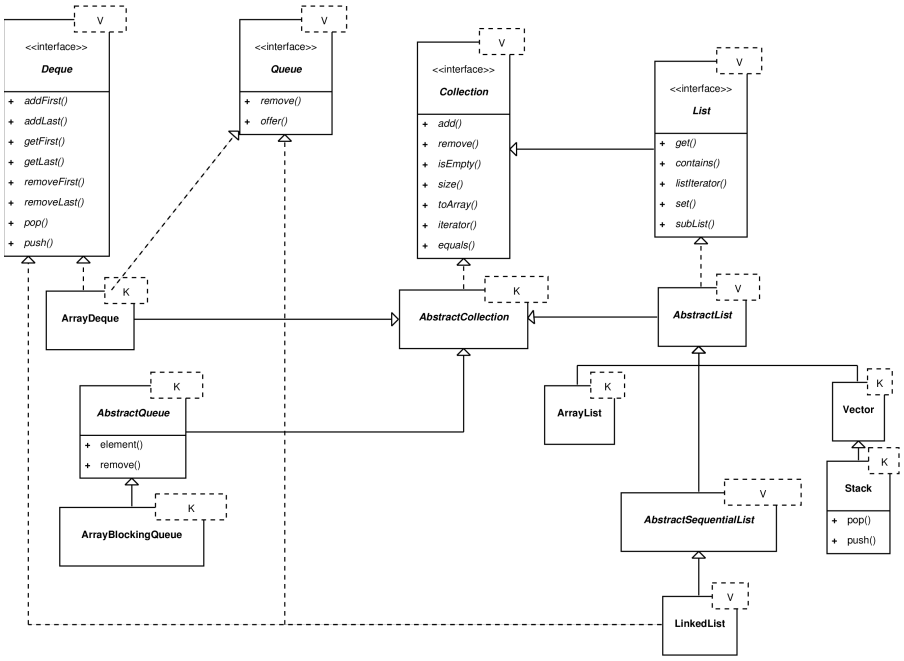


Diagrama de clases correspondiente a las estructuras de datos lineales de Java