**Problema 61:**

En la clase **DoublyLinkedOrderedList <T>** agrega el método *combina*, cuya firma aparece más abajo, que reciba una lista ordenada de forma ascendente y **combine** los valores que contiene con los de la lista ordenada actual (la cual suponemos que está ordenada de forma ascendente). Se desea que el método sea destructivo (es decir, después de ejecutar el método la lista actual queda modificada de tal forma que termina teniendo los valores que antes tenía además de los valores de la lista que se recibió como parámetro, todos ordenados de forma ascendente de manera **combinada**). Para no desperdiciar memoria se desea utilizar los nodos ya existentes en las dos listas (es decir, no se permite hacer copias de valores o nodos para producir la lista **combinada**). La lista que se recibió como parámetro también quedará alterada por la operación, pero no nos importa. Utiliza el método *toString* (aplicado antes y después de cada combinación) para demostrar que el método *combina* funcionó correctamente.

Las listas, la interna y la del parámetro, deben tener al menos un elemento.

Nota que en cualquier lista ordenada (tanto las originales como la que resulta de combinarlas) se permite que haya valores repetidos y no se desea perder ningún valor de ninguna de las listas originales al generar la lista combinada.

**public void combina(** ***DoublyLinkedOrderedList<T> lista* )**

El encabezado ya se encuentra escrito en la clase *DoublyLinkedOrderedList*, del paquete **Problema61DblLnkdLists**.

El método *main(…)* de la clase **Problema61Main**, compila y ejecuta correctamente, obviamente, sin que el método *combina(…)* tenga algún efecto. Viene un solo ejemplo de uso y prueba con una lista **DoublyLinkedOrderedList <Integer>**, **>**, agregue los ejemplos extras que aseguren el funcionamiento de su codificación.

Alguna posibles situaciones de las listas

+++++++++++++++

lista1 = {10, 20, 30}; Lista2 = {1, 2, 3}

lista1.combina(lista2);

Resultante lista1 = {1, 2, 3, 10, 20, 30}

+++++++++++++++

Lista3 = {10, 20, 30}; Lista4 = {40, 50, 60}

Lista3.combina(lista4);

Resultante lista3 = {10, 20, 30, 40, 50, 60}

+++++++++++++++

Lista5 = {10, 20, 30}; Lista6 = {15, 25, 35}

Lista5.combina(lista6);

Resultante lista5 = {10, 15, 20, 25, 30, 35}

+++++++++++++++

Lista7 = {10, 20, 30}; Lista8 = {5, 20, 24, 27}

Lista7.combina(lista8);

Resultante lista7 = {5, 10, 20, 20, 24, 27, 30}

+++++++++++++++

Y otras posibles situaciones.

**Problema 62:**

En la clase **Problema62Main** codifica el método estático, cuya firma aparece más abajo, que genere y regrese una lista ordenada con todas las ***revistas de divulgación*** que tengan un **tiraje mayor a *total*** y un **precio menor a *valprecio***. Tu solución debe ser recursiva.

La lista *material* debe tener al menos un elemento.

**public static DoublyLinkedOrderedList<**Divulgacion**> obtieneListaRevistas**

**( *DoublyLinkedOrderedList<****MaterialBibliografico****> material,***

***int total, double valprecio* )**

El encabezado ya se encuentra escrito en la clase **Problema62Main**, del paquete **Problema62DblLnkdLists**

El método *main(…)* de la clase **Problema62Main**, compila y ejecuta correctamente, obviamente, sin que el método *obtieneListaRevistas(…)* tenga algún efecto. Viene un solo ejemplo de uso y prueba con una lista **DoublyLinkedOrderedList <Integer>**, agregue los ejemplos extras que aseguren el funcionamiento de su codificación..

La jerarquía de clases de MaterialBibliografico es el siguiente:

|  |  |  |
| --- | --- | --- |
|  | MaterialBibliográfico  + |  |
| + + + + + + + + | + + + + + + | + + + + | + + + + + + + + + |
| + |  | + |
| Revista  + + + + + + + + + + |  | Libro |
| Científica Divulgación |  |  |