

PRÁCTICA A004.- OPERACIONES RECURSIVAS CON LISTAS

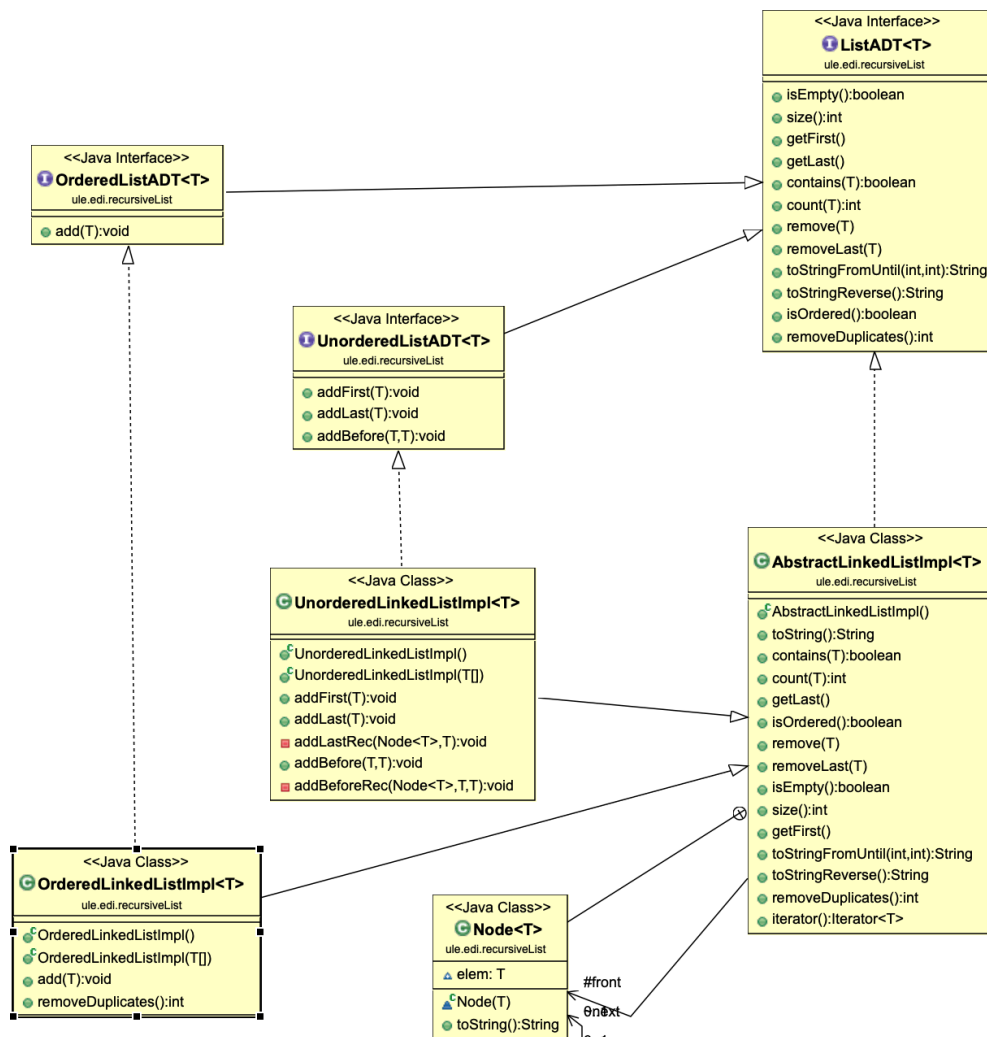
En esta práctica se trabajarán las **operaciones recursivas con listas**, por lo que habrá que implementar una serie de operaciones sobre listas simplemente enlazadas, la mayoría de ellas **DE FORMA RECURSIVA**.

La estructura de datos utilizada es una lista **simplemente enlazada** que viene definida por : **front**: referencia al primer nodo de la lista.

LAS ESTRUCTURAS DE DATOS NO SE PUEDEN MODIFICAR NI AÑADIR NUEVOS ATRIBUTOS.

PASOS PARA LA REALIZACIÓN DE LA PRÁCTICA:

1. **Descargar el proyecto ed_a004_2020** de la página de la asignatura en agora.unileon.es.
2. **Importar** dicho proyecto en Eclipse : Import... **General.....Existing projects into Workspace...** Select archive file... (indicar el archivo ZIP descargado)
3. El proyecto ed-a004-2020 contiene el paquete ule.edi.recursiveList que se compone de 3 interfaces, 3 clases de implementación y 2 clases de tests.



- **INTERFACES:** Los interface incluidos en el proyecto no se pueden modificar:
 - El interfaz **ListADT.java**: define varias operaciones para una lista simplemente enlazada:

public interface ListADT<T> extends Iterable<T>{

```
/**
 * Indica si esta lista está vacía
 * @return true si no contiene elementos
 */
boolean isEmpty();
```

```
/**
 * RECURSIVO
 * Devuelve el número total de elementos en esta lista.
 * Ejemplo: Si una lista l contiene (A B C B D A B ): l.size() -> 7
 * @return número total de elementos en esta lista
 */
int size();
```

```
/**
 * Devuelve el primer elemento de la lista. Si una lista l contiene (A B C D E ): l.getFirst -> A
 * @return primer elemento de la lista
 * @throws EmptyCollectionException si la lista es vacía
 */
T getFirst() throws EmptyCollectionException;
```

```
/**
 * RECURSIVO
 * Devuelve el último elemento de la lista. Si una lista l contiene (A B C D E ): l.getLast -> E
 * @return elem : último elemento de la lista
 * @throws EmptyCollectionException si la lista es vacía
 */
T getLast() throws EmptyCollectionException;
```

```
/**
 * RECURSIVO
 * Indica si el elemento está en esta lista.
 * Devuelve true si al menos existe una instancia del elemento dado en esta lista (es decir,
 * un elemento 'x' tal que x.equals(element) y false en caso contrario.
 *
 * @param element elemento a buscar en esta lista
 * @return true/false según el resultado
 * @throws NullPointerException el elemento indicado es null
 */
boolean contains(T element);
```

```
/**
 * RECURSIVO
 * Devuelve el número total de elementos iguales al pasado como parámetro.
 * Ejemplo: Si una lista l contiene (A B C B D A B): l.count("A") -> 2. l.count("B") -> 3
 * @param element elemento a contar sus apariciones
 * @return número total de elementos en esta lista
 * @throws NullPointerException el elemento indicado es null
 */
int count(T element);
```

```
/**
 * RECURSIVO
 * Elimina la primera aparición del elemento y devuelve el elemento.
 * Si una lista l contiene (A B C B D A B ): <br>
 * l.remove("A") -> A, dejando la lista (B C B D A B )
 * l.remove("B") -> B, dejando la lista (A C B D A B )
 * l.remove("Z") -> NoSuchElementException
 *
 * @param element elemento a eliminar.
 * @throws NullPointerException si element es null
 * @throws EmptyCollectionException si la lista está vacía.
 * @throws NoSuchElementException si element no está en la lista.
 */
T remove(T element) throws EmptyCollectionException;
```

```
/**
 * RECURSIVO
 * Elimina la última aparición del elemento y devuelve el elemento.
 * Si una lista l contiene (A B C B D A B ):
 * l.removeLast("A") -> A, dejando la lista (A B C B D B )
 * l.removeLast("B") -> B, dejando la lista (A B C B D A )
 * l.removeLast("Z") -> NoSuchElementException
 *
 * @param element elemento a eliminar
 * @throws NullPointerException si element es null
 * @throws NoSuchElementException si element no está en la lista.
 * @throws EmptyCollectionException si la lista está vacía.
 */
T removeLast(T element) throws EmptyCollectionException;
```

```
/**
 * RECURSIVO
 * Devuelve el toString de la sublista formada por los elementos situados entre las posiciones
 * from hasta until incluidas.
 * Si until > size() se muestra hasta el final de la lista.
 * Si from > size() se muestra () (toString de la lista vacía)
 * Ejemplos:
 * l1=(A B C D E ) ; l1.toSringFromUntil(1,3) -> (A B C )
 * l1=(A B C D E ) ; l1.toSringFromUntil(3,10) -> (C D E )
 * l1=(A B C D E ) ; l1.toStringFromUntil(10,20) -> ()
```

```
* @param from posición desde la que se empieza a considerar la lista (incluida)
* @param until posición hasta la que se incluyen elementos (incluida)
* @return String de la sublista de los elementos en el rango establecido por los dos parámetros
* @throws IllegalArgumentException si from o until son <=0 ; o si until < from
*/
public String toStringFromUntil(int from, int until);
```

```
/**
 * RECURSIVO
 * Crea y devuelve un String con el contenido de la lista en orden inverso.
 * Si esta lista es vacía devuelve el toString() de la lista vacía -> ().
 * Ejemplo: Si una lista l contiene (A B C) : l.toStringReverse() -> (C B A)
 * @return recorrido inverso de la lista (desde el final al principio)
 */
public String toStringReverse();
```

```
/**
 * RECURSIVO
 * Devuelve true si la lista está ordenada de menor a mayor
 * La lista vacía está ordenada.
 * @return true si los elementos están ordenados de menor a mayor; false en caso contrario.
 */
public boolean isOrdered();
```

```
/**
 * RECURSIVO
 * Elimina todos los duplicados de cada elemento, dejando solamente una aparición de cada
 * elemento y devuelve el número de elementos eliminados.
 * Si una lista l =(A B C B D A B) : l.removeDuplicates() -> 3 , dejando la lista (A B C D)
 * Si una lista l contiene (A B C) : l.removeDuplicates() -> 0, dejando la lista (A B C)
 * l.removeDuplicates("Z") -> NoSuchElementException
 *
 * @return el número de elementos que elimina
 * @throws EmptyCollectionException si la lista está vacía.
 */
public int removeDuplicates() throws EmptyCollectionException; }
```

- El interfaz **UnorderedListADT.java**: define varias operaciones para una lista simplemente enlazada **NO ORDENADA**:

```
public interface UnorderedListADT<T> extends ListADT<T>{
```

```
/**
 * Añade un elemento como primer elemento de la lista
 * Si una lista l = (A B D) y hacemos l.addFirst("C"), la lista quedará (C A B D)
 * @param element el elemento a añadir
 * @throws NullPointerException si element es null
 */
void addFirst(T element);
```

```
/**
 * RECURSIVO
 * Añade un elemento como último elemento de la lista.
 * Si una lista l=(A B D ) y hacemos l.addLast("C") , la lista quedará (A B D C)
 * @param element el elemento a añadir
 * @throws NullPointerException si element es null
 */
void addLast(T element);
```

```
/**
 * RECURSIVO
 * Añade un elemento delante de la primera aparición del elemento pasado como
 * 2º parámetro desplazando los elementos que estén a partir de ese elemento.
 * Si una lista l=(A B C B ) : l.addBefore("D", "B") ; la lista quedará (A D B C B ).
 *
 * @param element el elemento a añadir
 * @param target el elemento delante del cual insertará element
 * @throws NullPointerException si element o target son null
 * @throws NoSuchElementException si target no está en la lista
 */
void addBefore(T element, T target); }
```

- El interfaz **OrderedListADT.java**: define varias operaciones para una lista simplemente enlazada **ORDENADA**:

```
public interface OrderedListADT<T> extends ListADT<T>{
```

```
/**
 * Añade un elemento como de forma ordenada en la lista.
 * Si una lista l contiene (A B D ) y hacemos l.add("C") la lista quedará (A B C D )
 * @param element el elemento a añadir
 * @throws NullPointerException si element es null
 */
void add(T element)
```

- **CLASES DE IMPLEMENTACIÓN:**

- **AbstractLinkedListImpl<T>**: clase abstracta que implementa el interface ListADT<T> .

En esta clase el alumno debe implementar todas las operaciones del interface ListADT.

- **UnorderedLinkedListImpl<T>**: clase que hereda de la abstracta e implementa el interface UnorderedListADT<T>. Representa la clase Lista no ordenada.

En esta clase el alumno debe implementar todas las operaciones del interface UnorderedListADT.

- **OrderedLinkedListImpl<T>**: clase que hereda de la abstracta e implementa el interface **OrderedListADT<T>** . Representa la clase lista ordenada. (Los elementos de esta clase deben ser comparables)

En esta clase el alumno debe implementar la única operación del interface OrderedListADT<T>, que es el método add, que inserta un elemento de forma ordenada en la lista (el orden será de menor a mayor).

Además debe redefinir (override) el método removeDuplicates para que sea más eficiente al tener en cuenta que los elementos están ordenados y por tanto los duplicados estarán contiguos.

- **CLASES DE TESTS:**
 - **UnorderedLinkedListTests:** **En esta clase el alumno** incluirá todos los tests necesarios para probar toda la funcionalidad de la clase **AbstractLinkedListImpl** y la clase **UnorderedLinkedListImpl**.
 - **OrderedLinkedListTests:** **En esta clase el alumno** incluirá todos los tests necesarios para probar toda la funcionalidad de la clase **OrderedLinkedListImpl**.

Los tests deben estar basados en aserciones que realicen las comprobaciones automáticamente (no deben tener `System.out.println()` que requieran consultar la consola para saber si funcionan los tests o no).

En el proyecto hay un **paquete model** en el que está la clase **Person**, **en el que el alumno debe completar :**

- **el método equals (dos personas son iguales si son iguales sus nifs).**
- **El método compareTo: (Las personas se comparan por la edad). Este método debe devolver la diferencia de edad entre this y o.**
- **Se valorará la cobertura total de los test implementados. Se buscará el 100% en la cobertura de los archivos de implementación: AbstractLinkedListImpl, UnorderedLinkedListImpl y OrderedLinkedListImpl**

Se deberá entregar en agora.unileon.es la versión final de la práctica. Para ello habrá que exportar el proyecto edi-a004-2020 como zip (Export... General... Archive File).

IMPORTANTE: Utilizar JUNIT 4.

FECHA LIMITE DE ENTREGA: 15 de Mayo de 2020 a las 23:59