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, muchas de ellas **DE FORMA RECURSIVA.**

La estructura de datos utilizada es una lista simplemente enlazada que viene definida por :

- **header**: referencia al primer elemento de la lista.

PASOS PARA LA REALIZACIÓN DE LA PRÁCTICA:

1. **Descargar el proyecto edi_a004_2019** de la página de la asignatura en agora.unileon.es.
2. **Importar** dicho proyecto en Eclipse : Import… **General**…..**Existing proyects into Workspace**… Select archive file… (indicar el archivo ZIP descargado)
3. El proyecto edi_api_2018 no hay que modificarlo. El paquete ule.edi.recursive contiene :
   a. El interfaz **SingleLinkedList.java**: que define la operación addLast(T element) que hay que implementar de **FORMA RECURSIVA**:

```
public interface SingleLinkedList<T> extends Iterable<T> {
        /**
         * Implementar de forma RECURSIVA
         */
        public void addLast(T element);
}
```

   b. La clase abstracta **AbstractSingleLinkedListImpl<T>** que implementa el interface SingleLinkedList<T> donde se definen las operaciones abstractas a implementar en el proyecto edi-a004-2019:

```
public abstract class AbstractSingleLinkedListImpl<T> implements SingleLinkedList<T> {
/**
         * indica si la lista está vacía.
         *  @return true si la lista es vacía, false en caso contrario
         */
        public abstract boolean isEmpty();

        /**
         *     Implementar de forma RECURSIVA
         *
         * Devuelve el número de elementos de la lista.
         *  @return numero de elementos de la lista
         */
        public abstract int size();

        /**
         * Inserta un elemento como primero.
         * @param element a insertar
         */
        public abstract void addFirst(T element);

        /**
         *    Implementar de forma RECURSIVA
         *
         * Inserta un elemento como último.
         *
         * @param element a insertar
         */
        public abstract  void addLast(T element);


        /**
         *     Implementar de forma RECURSIVA
         *
         * Inserta el elemento en la posición p, desplazando los elementos a partir de esa posición.
         * Si la lista tiene menos de n elementos lo insertará como último elemento.
         *
         * Si la lista era [A, B, C] :
         *       lista.addAtPos("Z", 1) dejará la lista como [Z, A, B, C].
```

```
         *       lista.addAtPos("Z", 3) dejará la lista como [A, B, Z, C].
         *       lista.addAtPos("Z", 5) dejará la lista como [A, B, C, Z].
         *
         * @param element a insertar
         * @param pos posicion en la que se insertará el elemento, desplazando los siguientes
         * @throw IllegalArgumentException si p<=0
         */
        public abstract void addAtPos(T element, int p);

        /**
         *   Implementar de forma RECURSIVA
         *
         * inserta n veces el elemento al final de la lista.
         * Si lista=[A, B, C], lista.addNTimes("Z", 4) dejará la lista como: [A, B, C, Z, Z, Z, Z]
         *
         * @param element a insertar
         * @param p posicion en la que se insertará el elemento, desplazando los siguientes
         * @throw IllegalArgumentException si N<0
         */
        public abstract void addNTimes(T element, int n);



        /**
         *    Implementar de forma RECURSIVA
         *
         * Indica la posición donde se encuentra la primera aparición de elem desde el principio de la lista
(las posiciones empiezan en 1).
         * Dispara la excepción NoSuchElementException si no se encuentra el elemento en la lista.
         *
         * @param elem  el elemento a buscar
         * @return la posición que ocupa el elemento en la lista
         * @throws NoSuchElementException si no se encuentra el elemento en la lista.
         */
        public abstract int indexOf(T elem);


        /**
         *       Implementar de forma RECURSIVA
         *
         * Elimina el último elemento de la lista.
         *
         * @return el elemento que es eliminado
         * @throws EmptyCollectionException si la lista está vacía
         */
        public abstract T removeLast() throws EmptyCollectionException ;

        /**
         *   Implementar de forma RECURSIVA
         *
         * Elimina la última aparición del elemento.
         * Si la lista es vacía dispara la excepción EmptyCollectionException.
         *
         * Si lista=[A, C, B, C, D, C]
         *              lista.removeLast("C") dejará a lista=[A, C, B, C, D]
         *
         * @param elem  el elemento a eliminar
         * @return el elemento que es eliminado
         * @throws EmptyCollectionException si la lista está vacía
         * @throws NoSuchElementException si no se encuentra el elemento en la lista
         */
        public abstract T removeLast(T elem) throws EmptyCollectionException;


        /**
         *
         *        Implementar de forma RECURSIVA
         *
         * Devuelve la lista inversa de la lista actual.
         * Deja la lista actual sin modificar.
         * Por ejemplo, si la lista era [A, B, C], la lista devuelta será [C,B,A]
         *
         */
        public abstract AbstractSingleLinkedListImpl<T> reverse();
```

```java
	/**
	 *
	 *        Implementar de forma RECURSIVA
	 *
	 * Indica a partir de qué posición se encuentra la sublista pasada como parámetro en la lista actual
	 *       o -1 si no se encuentra.
	 * Si part es vacía devuelve 1
	 *
	 * Ejemplos:
	 *
	 *  [A, B, A, B, C], con part=[B, A, X], devolvería -1
	 *  [A, B, A, B, C], con part=[B, A], devolvería 2
	 *
	 *  [A, B, A, B], con part=[A, B], devolvería 1
	 *
	 *  [A, B, A, B, C, X, A], con part=[B, C, X], devolvería 4
	 *
	 * @param part lista a comprobar si es sublista de la actual
	 * @return posición a partir de la que se encuentra la sublista en la lista actual
	 *
	 */
	public abstract int isSubList(AbstractSingleLinkedListImpl<T> part) ;

//	estructura de datos y métodos ya implementados

	static class Node<G> {

		public Node(G element) {
			this.content = element;
			this.next = null;
		}

		G content;

		Node<G> next;
	}

	protected Node<T> header;


	@Override
	public String toString() {
		if (header != null) {
			StringBuffer rx = new StringBuffer();
			rx.append("[");
			Node<T> i = header;
			while (i != null) {
				rx.append(i.content);
				rx.append(", ");
				i = i.next;
			}
			rx.delete(rx.length() - 2, rx.length());
			rx.append("]");

			return rx.toString();
		} else {
			return "[]";
		}
	}
```

2. En el proyecto edi_a004_2019 habrá que completar los métodos de la clase **SingleLinkedListImpl**<T> que hereda de la clase abstracta AbstractSingleLinkedListImps<T>

3. Al igual que en el resto de prácticas, a la vez que se van desarrollando los métodos recursivos anteriores se deben crear los correspondientes métodos de prueba JUnit 4 para ir comprobando su correcto funcionamiento en la clase **`SingleLinkedListImpl<T>`.**

4. Además **se deberá entregar en agora.unileon.es la versión final de la práctica** (proyecto exportado como zip) .

<u>NOTA IMPORTANTE</u>: NO SE PUEDEN UTILIZAR LAS ESTRUCTURAS DE DATOS DEL API COLLECTIONS DE JAVA (HAY QUE UTILIZAR LAS ESTRUCTURAS DE DATOS INDICADAS EN LAS CLASES DEL PROYECTO edi-a004-2019)

FECHA LIMITE de entrega de la práctica A004-2019: 4 de Mayo de 2019 a las 23:55