PRÁCTICA 4B: PRUEBAS DE SOFTWARE

1. Objetivos

- Aprender a diseñar casos de prueba unitarios aplicando técnicas de caja negra y caja blanca para un sistema software orientado a objetos.
- Aprender a ejecutar pruebas unitarias utilizando un framework de pruebas, en este caso formado por las herramientas *JUnit*.

2. Actividades

Para una futura integración con el resto de la aplicación de la práctica 3, es necesario implementar el proceso de pruebas unitarias de la estructura de datos `ListaOrdenadaAcotada`. Para las pruebas de caja negra se deben aplicar las técnicas de partición equivalente y AVL y para las pruebas de caja blanca se requiere cobertura de sentencias y de decisión/condición.

La documentación de la clase `ListaOrdenadaAcotada` es la siguiente:

```
/**
 * Clase que implementa el DTO IListaOrdenadaAcotada.
 * Su capacidad máxima se asigna a través del constructor.
 */
public class ListaOrdenadaAcotada<E extends Comparable<E>>
       implements IListaOrdenadaAcotada<E> {

   public final static int MAX_DEFAULT = 10;

    /**
     * Constructor al que se le pasa la capacidad
     * máxima de la lista
     * @throws NegativeArraySizeException si max es negativo
     */
    public ListaOrdenadaAcotada(int max) {}

}
```

Como se puede observar, la clase a probar, `ListaOrdenada`, cuyo código se encuentra disponible en Moodle, implementa la interfaz `IListaOrdenadaAcotada`, cuya documentación se muestra en el anexo.

La interfaz `IListaOrdenadaAcotada` forma parte de una librería desarrollada como proyecto Maven, con los siguientes datos:

- groupId = es.unican.is2
- artifactId = AbstractDataTypes
- versión = 1.0

Esta librería se encuentra disponible en Moodle en forma de archivo .jar y deberá ser debidamente instalado en el sistema para poder proceder a su uso (ver transparencia 16 del Seminario de Maven).

Los pasos a seguir para llevar a cabo el proceso de prueba son los siguientes:

i. Diseñar casos de prueba de caja negra basándose en la interfaz de la clase, `IListaOrdenadaAcotada`.

ii. Crear un proyecto Maven e implementar en él los casos de prueba definidos usando JUnit.
iii. Ejecutar los casos de prueba y detectar posibles errores.
iv. Corregir los defectos encontrados.
Importante: No corregir los errores antes de ejecutar las pruebas. La ejecución de la clase de prueba debe poner de manifiesto en qué métodos hay errores.
v. Ejecutar pruebas de caja blanca comprobando la cobertura alcanzada.
vi. En caso de que la cobertura no sea suficiente, diseñar nuevos casos de prueba.

3. Criterios de Evaluación y Aclaraciones

Se deberá entregar un fichero comprimido que contenga los siguientes elementos:

- Proyecto Maven completo, con todas las clases involucradas, debidamente exportado.
- Un informe que resuma los casos de prueba diseñados.

Patricia López Martínez
Juan Rivas Concepción
Adolfo Garandal

ANEXO. INTERFAZ IListaOrdenadaAcotada

```
/**
 * ADT ListaOrdenadaAcotada
 * Estructura de datos que almacena datos ordenados de acuerdo
 * a su orden natural y tiene capacidad limitada. No admite nulos.
 * @param <E> tipo de los elementos almacenados en la lista
 */
public interface IListaOrdenadaAcotada<E extends Comparable<E>> {

       /**
        * Retorna el elemento que ocupa la posicion indicada
        * @param indice Indice del elemento
        * @return Elemento que ocupa la posicion indice
        * @throws IndexOutOfBoundsException si el indice es incorrecto
        */
    public E get(int indice);

    /**
     * Inserta un elemento en la posicion que le corresponde
     * de acuerdo a su orden natural (usando el compareTo)
     * @param elemento a insertar
     * @throws Lanza IllegalStateException si el elemento no cabe
     * @throws Lanza NullPointerException si el elemento es nulo
     */
    public void add(E elemento);

    /**
     * Elimina el elemento que ocupa la posicion indicada
     * @param indice Posicion del elemento a eliminar
     * @return Elemento que ocupaba la posicion indicada
     * Lanza IndexOutOfBoundsException si el indice es incorrecto
     */
    public E remove(int indice);

    /**
     * Retorna el numero de elementos en la lista
     * @return Numero de elementos
     */
    public int size();

    /**
     * Vacia la lista
     */
     public void clear();

     /**
      * Retorna la capacidad maxima de la lista
      */
     public int capacidadMaxima();

}
```