Sesión 27: Ejercicio cadenas enlazadas

`LinkedList<T>`

Programación 2

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

9 de Abril 2019

Normas

- El ejercicio debe realizarse **individualmente**.
- El ejercicio dura **1 día**.
- **Lee todo el enunciado** antes de empezar.
- La entrega deberá realizarse a través de la **tarea Moodle** indicada por el profesor: "Ejercicio cadenas enlazadas (23 de mayo)".
- Junto a este PDF habrás encontrado un **fichero *.zip*** con esquemas de código que te ayudarán a realizar el ejercicio. Mi recomendación es que lo descomprimas en un directorio (por ejemplo `sesion27`) y trabajes en él. Al descomprimirlo verás dos directorios (`./src` y `./bin`), dentro de `./src` podrás encontrar código de apoyo que te servirá como punto de partida.
- **Tendrás que entregar un único fichero:** `LinkedList.java`. La primera línea del fichero deberá ser un comentario con el **nombre del autor** siguiendo este formato: *`//Autor: Herranz Nieva, Ángel`*

Ayuda

A modo de ayuda te cuento cómo puedes compilar y ejecutar desde la línea de comandos. Asumo que has descomprimido el *.zip* en un directorio y que abres una consola y te cambias hasta dicho directorio[1].

- **Compilar**[2]

```
$ javac -Xlint:unchecked -d bin -cp src -s src src/TestLinkedList.java
```

- **Ejecutar** (con *aserciones* habilitadas: `-ea`):

```
$ java -ea -cp bin TestLinkedList
```

[1] Los comandos han sido ejecutados en Unix, si usas Windows tendrás que utilizar "\" en vez de "/" en los nombres de los ficheros

[2] `javac` ya sabe compilar todo lo que depende del fichero indicado

Ejercicio 1. `Node`

Empieza explorando la implementación de la clase que representa nodos de cadenas enlazada. En esencia es la misma clase que la clase `Nodo` vista en las sesiones de cadenas enlazadas.

La clase que vas a tener que usar se llama `Node` (como *nodo* pero en inglés). La diferencia es que tanto el dato que encapsula como la referencia al siguiente quedan ocultas (**`private`**). A cambio dispones de operaciones para acceder y modificar el dato así como para acceder y modificar el siguiente.

Analiza la implementación en `src/node/Node.java` en el código de apoyo.

Ejercicio 2. Implementa la clase `list.LinkedList`:

```
public class LinkedList<E> implements ListInterface<E>
{
  private Node<E> head;
  private int nElems;

  public LinkedList() {
    head = null;
    nElems = 0;
  }

  // TODO: implementar el API completo
}
```

Como puedes ver, además de una cadena enlazada (`head`), tu clase deberán mantener un entero con el número de elementos en la cadena (`nElems`) para evitar tener que recorrer la cadena cuando se quiera conocer el tamaño de la lista.

El fichero `src/TestLinkedList.java` contiene unos tests bastante fuertes que tu implementación deberá pasar antes de que realices la entrega. Recuerda que para ejecutar los tests desde la línea de comandos tendrás que ejecutar el siguiente mandato:

```
$ java -ea -cp bin TestLinkedList
```

Lo que debe hacer cada operación lo tienes descrito en el interfaz que `LinkedList` tiene que implementar:

```
package list;

/**
 * Definición de un TAD Lista
 *
 * @author jramirez
 *
 */
public interface ListInterface<E> {

  /**
   * Coloca un nuevo elemento en la posición insertIndex
   *
   * <br><B>PRE:</B> insertIndex EN {0..size()}
   * <br><B>POST:</B> devuelve la lista this con element en la pos insertIndex
   * y los elementos que antes estaban en pos &gt;=insertIndex, ahora en pos+1.
   *
   * @throws IndexOutOfBoundsException
   */
  public void add(int insertIndex, E element) throws IndexOutOfBoundsException;

  /**
   * Lectura indexada de una posición de la lista
```

```
     *
     * <br><B>PRE:</B> insertIndex EN {0..size()-1}
     * <br><B>POST:</B> devuelve una ref al elemento que está en la pos index.
     *
     * @throws IndexOutOfBoundsException
     */
    public E get(int getIndex) throws IndexOutOfBoundsException;

    /**
     * No. de elementos en la lista
     *
     * <br><B>PRE:</B> cierto
     * <br><B>POST:</B> devuelve el no. de elems que hay en la lista.
     */
    public int size();

    /**
     * Escritura indexada en una posición de la lista
     *
     * <br><B>PRE:</B> insertIndex EN {0..size()-1}
     * <br><B>POST:</B> coloca element en la posición insertIndex de la lista destruyendo
     * el elemento que había en esa posición.
     *
     * @throws IndexOutOfBoundsException
     */
    public void set(int insertIndex, E element) throws IndexOutOfBoundsException;

    /**
     * Posición de un elemento dentro de la lista
     *
     * <br><B>PRE:</B> Cierto
     * <br><B>POST:</B> devuelve una ref al primer elemento de la lista
     * que es igual a search (equals), o -1 si no existe ningún elemento igual a search.
     *
     */
    public int indexOf(E search);

    /**
     * Extracción de un elemento de la lista dada su posición
     *
     * <br><B>PRE:</B> removalIndex EN {0..size()-1}
     * <br><B>POST:</B> extrae el elemento que está en la pos removalIndex.
     *
     * @throws IndexOutOfBoundsException
     */
    public void removeElementAt(int removalIndex) throws IndexOutOfBoundsException;

    /**
     * Extracción de un elemento de la lista dado un elemento igual (equals)
     *
     * <br><B>PRE:</B> cierto
     * <br><B>POST:</B> extrae el primer elemento que sea igual a element (equals) y devuelve cierto,
     * si existe. Y si no existe, devuelve falso.
     *
     */
    public boolean remove(E element);

}
```