UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Práctica 3: Estructuras de datos básicas con memoria dinámica

Ejercicio 1

En este ejercicio vamos a utilizar algunas de las clases disponibles en las bibliotecas del lenguaje Java para almacenar colecciones de objetos. Estas clases se encargan de realizar la reserva y gestión de memoria necesaria para la estructura de datos correspondiente.

a) Reescribe tu implementación de la clase `Agenda` del ejercicio 9 de la Práctica 2 para reemplazar el vector de tareas por un objeto de la clase `ArrayList` y modificar el cuerpo[1] de los métodos que lo requieran para adaptarse a la nueva representación interna. De todos los métodos disponibles en la clase `ArrayList`, utiliza solo aquellos declarados en la interfaz `List`.

   A la hora de reescribir el método `añadir`, te aconsejamos usar un bucle `for` avanzado que recorra la lista de tareas y una variable que indique la posición actual en esa lista.

b) Sustituye el objeto de la clase `ArrayList` por un objeto de la clase `LinkedList`. Al haber utilizado solamente métodos de la interfaz `List`, el resto de tu código debería funcionar perfectamente.

Ejercicio 2

Considera la siguiente interfaz `ListaCadenas`:

```
public interface ListaCadenas {
    boolean add(String s);
    void add(int i, String s);
    void clear();
    String get(int i);
    int indexOf(String s);
    int lastIndexOf(String s);
    boolean isEmpty();
    String remove(int i);
    boolean remove(String s);
    String set(int i, String s);
    int size();
}
```

Escribe la clase `ListaCadenasEnlaceSimple` que implemente la interfaz `ListaCadenas`. Para ello, debes utilizar nodos enlazados con un enlace al siguiente elemento. La lista debe tener como atributos únicamente la talla y una referencia al primer nodo.

A continuación, se describe el comportamiento de cada uno de los métodos de la interfaz:

- `boolean add(String s)`

  Añade la cadena `s` al final de la lista. Devuelve como resultado `true` para indicar que la operación se realizó correctamente.

[1] Recuerda que al no modificar la cabecera de los métodos públicos de la clase `Agenda`, los programas que hacían uso de la misma deben seguir funcionando.

- `void add(int i, String s)`

  Inserta la cadena `s` en la posición `i` de la lista. El elemento en la posición dada (si existe) y todos los siguientes quedan desplazados una posición a la derecha en la lista. Si la posición `i` no es válida (`i < 0` o `i > size()`), se debe lanzar la excepción `IndexOutOfBoundsException`.

- `void clear()`

  Elimina todos los elementos en la lista, es decir, la lista quedará vacía.

- `String get(int i)`

  Devuelve el elemento en la posición `i`. Si la posición `i` no es válida (`i < 0` o `i >= size()`), se debe lanzar la excepción `IndexOutOfBoundsException`.

- `int indexOf(String s)`

  Devuelve la posición de la primera ocurrencia de la cadena `s` en la lista o -1 si la cadena `s` no aparece en la lista.

- `int lastIndexOf(String s)`

  Devuelve la posición de la última ocurrencia de la cadena `s` en la lista o -1 si la cadena `s` no aparece en la lista.

- `boolean isEmpty()`

  Devuelve `true` si la lista no contiene elementos o `false` en caso contrario.

- `String remove(int i)`

  Elimina el elemento en la posición `i` de la lista. Los elementos posteriores quedan desplazados una posición a la izquierda. Devuelve como resultado la cadena que se eliminó. Si la posición `i` no es válida (`i < 0` o `i >= size()`), se debe lanzar la excepción `IndexOutOfBoundsException`.

- `boolean remove(String s)`

  Elimina la primera ocurrencia de la cadena `s` en la lista, si esta existe. Si la cadena no aparece en la lista, esta no se modifica. Devuelve un valor lógico que indica si la lista ha sido modificada, es decir, si se ha borrado la cadena dada.

- `String set(int i, String s)`

  Sustituye la cadena en la posición `i` por la cadena `s`. Devuelve como resultado la cadena previa en la posición indicada. Si la posición `i` no es válida (`i < 0` o `i >= size()`), se debe lanzar la excepción `IndexOutOfBoundsException`.

- `int size()`

  Devuelve la cantidad de elementos en la lista.

Además de los métodos de la interfaz `ListaCadenas` debes implementar el método:

- `String toString()`

  Convierte un objeto de la clase en una cadena. Debes utilizar el formato:

  ```
  [elemento, elemento, ..., elemento]
  ```

Para comprobar el correcto funcionamiento de tu clase puedes utilizar las pruebas que tienes disponible en el *Aula Virtual*.

Ejercicio 3

Una manera de mejorar la eficiencia de los métodos que acceden a una posición dada de la lista es aumentar la clase con una referencia al último nodo y añadir a los nodos un enlace al nodo anterior. De este modo tenemos una lista doblemente enlazada. Cuando queremos acceder a la posición `i` de la lista, podemos comprobar si está en la primera o en la segunda mitad. Si está en la primera, avanzamos como hasta ahora. Si está en la segunda, comenzamos el recorrido en el último nodo y retrocedemos hasta la posición deseada. De esta forma, nunca tenemos que recorrer más de media lista.

Escribe la clase `ListaCadenasEnlaceDoble` que utilice la idea descrita en el párrafo anterior, es decir, debes utilizar nodos enlazados con enlaces a los elementos anterior y siguiente. La lista debe tener como atributos únicamente la talla, una referencia al primer nodo y una referencia al último nodo. Además de los métodos de la interfaz `ListaCadenas`, debes implementar el método `toString`.

Las pruebas del ejercicio anterior también deben funcionar en este caso.

Ejercicio 4

En este ejercicio vamos a resolver estos dos problemas prácticos:

a) la obtención de la palabra que aparece con más frecuencia en un libro, y

b) la obtención del ganador de la liga de fútbol a partir de los resultados de todos los partidos.

Ambos problemas son en apariencia muy diferentes, pero si los estudiamos de cerca no lo son en absoluto. En los dos casos podemos *asociar una cadena con una cantidad numérica*: en el primer problema podemos asociar cada palabra con su número de apariciones y en el segundo problema podemos asociar el nombre de un equipo de fútbol con su puntuación.

Consideremos ahora una estructura de datos en la que podemos almacenar una serie de parejas *cadena–cantidad* como las descritas. En esa estructura de datos, nunca aparecerá la misma cadena en dos parejas diferentes: en el primer problema, cada palabra del libro es única y en el segundo problema cada equipo de fútbol es único.

La estructura de datos que estamos definiendo recibe el nombre de *diccionario*, y las parejas que almacena reciben el nombre de *par clave–definición*. A esta estructura de datos se le asocia una serie de operaciones típicas, pero aquí solo vamos a considerar una variante adaptada a la resolución los dos problemas planteados.

Escribe la clase `Diccionario` que cumpla los siguientes requisitos:

- La serie de parejas *cadena–cantidad* se debe gestionar mediante una lista enlazada de nodos. Cada nodo debe contener el atributo `cadena` de tipo `String` y el atributo `cantidad` de tipo `int`. Puedes utilizar nodos con enlace simple o doble.
- Los nodos en la lista se mantendrán siempre ordenados lexicográficamente según su atributo `cadena`.
- No puede haber dos nodos cuyos atributos `cadena` sean iguales.

Los métodos públicos que debe proporcionar la clase son:

- `void añadir(String unaCadena, int unaCantidad)`

  Si existe algún nodo cuyo atributo `cadena` coincide con `unaCadena` entonces su `cantidad` se incrementa con `unaCantidad`. En caso contrario, se crea un nuevo nodo con `unaCadena` y `unaCantidad` y se enlaza en la lista en su posición.

- `String cadenaConMayorCantidad()`

  Devuelve la `cadena` correspondiente al nodo cuyo atributo `cantidad` sea mayor. En caso de empate entre las cantidades de dos nodos, devuelve la `cadena` de cualquiera de ellos. Si el diccionario está vacio, devuelve `null`.

- `int cantidad(String unaCadena)`

  Devuelve como resultado la `cantidad` asociada a `unaCadena`. Si `unaCadena` no existe en ningún nodo de la lista, devuelve como resultado `0`.

- `String toString()`

  Devuelve como resultado una cadena que representa el diccionario. Para facilitar la implementación de este método, te aconsejamos que definas el método `toString()` también en la la clase `Nodo`. Debes utilizar el formato:

  ```
  {cadena-->cantidad, cadena-->cantidad, ..., cadena-->cantidad}
  ```

Comprueba el correcto funcionamiento de la clase `Diccionario` con las pruebas que encontrarás en el *Aula Virtual*.

Resuelve ahora en ficheros diferentes los dos problemas planteados al principio:

a) Escribe el programa `PalabraMasFrecuente` que lea las palabras del fichero `DonQuijote.txt` (disponible en el *Aula Virtual*) y que, utilizando la clase `Diccionario`, muestre en la pantalla la palabra más frecuente y su número de apariciones.

Para realizar la lectura de las palabras de forma sencilla basta con que utilices el método `next()` de la clase `Scanner`. De esta manera, consideramos que las palabras son simples secuencias de caracteres separadas por espacios en blanco, tabuladores o retornos de carro. No hagas ningún tratamiento sobre las cadenas leídas con `next()`, en concreto, no elimines los signos de puntuación que puedan contener.

La salida que debe mostrar tu programa es:

```
La palabra más frecuente en DonQuijote.txt es «que» con 19429 apariciones.
```

Prueba ahora con el fichero `AlicesAdventuresInWonderland.txt`. (disponible en el *Aula Virtual*). En ese caso la salida que debe mostrar tu programa es:

```
La palabra más frecuente en AlicesAdventuresInWonderland.txt es «the» con 1530 apariciones.
```

b) Escribe el programa `GanadorLiga` que lea el fichero `liga2015-2016.txt` (disponible en el *Aula Virtual*). Este fichero contiene en cada línea el resultado de un partido de la liga de fútbol de la temporada 2015-2016[2] con el formato siguiente: el nombre del equipo local, el número de goles que ha marcado, el nombre del equipo visitante y el número de goles que ha marcado. Cada dato se separa del siguiente por un espacio en blanco. Los nombres de los equipos no contienen espacios en blanco. Las primeras líneas del fichero son:

```
Málaga 0 Sevilla 0
Deportivo_de_La_Coruña 0 Real_Sociedad 0
Espanyol 1 Getafe 0
...
```

[2] Datos reales de la temporada 2015-2016 obtenidos de `http://www.bdfutbol.com/es/t/t2015-16.html`.

Tu programa debe utilizar la clase `Diccionario` para calcular el ganador de la liga. El ganador es aquel equipo que consigue mayor puntuación. Para cada partido, los puntos se reparte de la siguiente manera: el equipo que ha marcado más goles obtiene 3 puntos y en caso de empate cada equipo obtiene 1 punto.

La salida que debe mostrar tu programa es:

```
Ganador liga 2015-2016: «Barcelona» con 91 puntos.
```