Caso Práctico EDA 2015-2016

Programa para la creación de un diccionario para un corrector ortográfico.

Un corrector ortográfico es utilizado por los editores de texto como una funcionalidad para analizar el texto y detectar los posibles errores ortográficos.

En este caso práctico **no vamos a desarrollar un corrector ortográfico**. Nuestro objetivo es mucho más concreto y nos limitaremos a desarrollar la(s) estructura(s) de datos necesaria(s) para almacenar el diccionario de palabras, que utilizaría un corrector ortográfico. En concreto, necesitamos desarrollar una estructura capaz de almacenar una colección de palabras. Además, para cada palabra será necesario almacenar algunas de sus posibles variaciones (posibles errores ortográficos de la palabra). En la parte 2, se explica con detalle el método para generar estas variaciones.

Parte 1:

Consiste en construir una estructura lineal adecuada para almacenar una colección de palabras. Dicha estructura se llamará **ListaPalabras.** En esta primera parte, por el momento sólo vamos a guardar las palabras y no sus posibles variaciones.

Nota: En ningún caso se permitirá utilizar un array ni ninguna de las clases del API de Java para implementar colecciones, como por ejemplo, *ArrayList*, *LinkedList*, etc. Es decir, el/la alumno/a deberá implementar su propio TAD que almacene la colección de palabras.

A continuación se describen los métodos de **ListaPalabras**:

- **void insertarPalabra(String p)**: el método recibe un objeto de tipo String y añade dicho objeto a la colección. La palabra debe ser almacenada en orden alfabético ascendente (de la ‘a’ a la ‘z’). Si la palabra ya existe, no debe ser insertada y el método deberá mostrar un mensaje informando que ya existe en la colección.

- **void eliminarPalabra(String p)**: recibe un objeto de tipo String y elimina dicha palabra de la colección.

- **int buscarPalabra(String p)**: el método recibe un objeto p de tipo String y comprueba si es una palabra de la colección. En caso de existir, devolverá la posición en la que se encuentra dentro de la colección. Recuerda que en Java, el

primer elemento de una colección siempre tiene índice 0. Si no existe, devolverá -1.

- **void consultarLetra(char c)**: dado una letra *c*, el método buscará en la colección todas las palabras que empiecen por dicha letra y las mostrará por pantalla. Si la letra c es el carácter ' ', el método deberá devolver todas las palabras almacenadas en la colección.

Además de la implementación de la estructura **ListaPalabras** y sus métodos, también será necesario completar los siguientes apartados:

1. Desarrolla una clase Test que te permita probar cada uno de los métodos de la estructura ListaPalabras.
2. Explicar brevemente cuál es el orden de complejidad (temporal) de cada método.

Parte 2:

El objetivo de esta parte del caso práctico es desarrollar un método que dada una palabra genere sus posibles variaciones ortográficas. Las variaciones ortográficas se consigue aplicando alguna de las siguientes operaciones:

- Inserción: consiste en añadir una letra a la palabra. El método deberá generar todas las posibles variaciones utilizando esta operación con cualquier letra y en cualquier posición.
- Borrado: consiste en borrar una letra de la palabra. El método deberá generar todas las posibles variaciones que se producen al eliminar cada una de sus letras.
- Modificación: consiste en modificar una letra de la palabra. El método deberá generar todas las posibles variaciones al considerar la modificación de cada una de las letras de la palabra.

Palabra original	**Algunas variaciones**	**Tipo de operación**
otra	**M**otra, otra**s,** o**p**tra, etc	Inserción
patata	Atata, ptata, paata, etc	Eliminación
mot**a**	Mot**o, c**ota, **r**ota, m**a**ta, m**b**ta, m**c**ta, etc	Modicación

Muchas de las variaciones que tu método generará no son palabras que existan en un diccionario. No te preocupes por ello. Nuestro objetivo no es generar palabras correctas, sino todas las posibles variaciones de una palabra dada.

Además, recuerda que cada una de las variaciones se obtiene aplicando una única operación. Así por ejemplo, dada la palabra "cota", el método no debería devolver las variaciones como "motas" o "patos" porque implican cada una esa esas variaciones implican dos o más operaciones.

Para almacenar todas estas posibles variaciones utilizaremos un objeto de la estructura ListaPalabras, implementado en la parte 1. De esta forma, el método deberá devolver un objeto de tipo ListaPalabras con todas las variaciones para la palabra p.

Veamos un ejemplo concreto. Supongamos que queremos calcular todas las posibles variaciones de la palabra **otra**. El conjunto de todas las posibles para la palabra "otra" serían:

AOTRA	ATRA	BOTRA	BTRA	COTRA	CTRA	DOTRA	DTRA	EOTRA	ETRA	FOTRA
FTRA	GOTRA	GTRA	HOTRA	HTRA	IOTRA	ITRA	JOTRA	JTRA	KOTRA	KTRA
LOTRA	LTRA	MOTRA	MTRA	NOTRA	NTRA	OARA	OATRA	OBRA	OBTRA	OCRA
OCTRA	ODRA	ODTRA	OERA	OETRA	OFRA	OFTRA	OGRA	OGTRA	OHRA	OHTRA
OIRA	OITRA	OJRA	OJTRA	OKRA	OKTRA	OLRA	OLTRA	OMRA	OMTRA	ONRA
ONTRA	OORA	OOTRA	OPRA	OPTRA	OQRA	OQTRA	ORA	ORRA	ORTRA	OSRA
OSTRA	OTA	OTAA	OTARA	OTBA	OTBRA	OTCA	OTCRA	OTDA	OTDRA	OTEA
OTERA	OTFA	OTFRA	OTGA	OTGRA	OTHA	OTHRA	OTIA	OTIRA	OTJA	OTJRA
OTKA	OTKRA	OTLA	OTLRA	OTMA	OTMRA	OTNA	OTNRA	OTOA	OTORA	OTPA
OTPRA	OTQA	OTQRA	OTR	OTRA	OTRAA	OTRAB	OTRAC	OTRAD	OTRAE	OTRAF
OTRAG	OTRAH	OTRAI	OTRAJ	OTRAK	OTRAL	OTRAM	OTRAN	OTRAO	OTRAP	OTRAQ
OTRAR	OTRAS	OTRAT	OTRAU	OTRAV	OTRAW	OTRAX	OTRAY	OTRAZ	OTRB	OTRBA
OTRC	OTRCA	OTRD	OTRDA	OTRE	OTREA	OTRF	OTRFA	OTRG	OTRGA	OTRH
OTRHA	OTRI	OTRIA	OTRJ	OTRJA	OTRK	OTRKA	OTRL	OTRLA	OTRM	OTRMA
OTRN	OTRNA	OTRO	OTROA	OTRP	OTRPA	OTRQ	OTRQA	OTRR	OTRRA	OTRS
OTRSA	OTRT	OTRTA	OTRU	OTRUA	OTRV	OTRVA	OTRW	OTRWA	OTRX	OTRXA
OTRY	OTRYA	OTRZ	OTRZA	OTSA	OTSRA	OTTA	OTTRA	OTUA	OTURA	OTVA
OTVRA	OTWA	OTWRA	OTXA	OTXRA	OTYA	OTYRA	OTZA	OTZRA	OURA	OUTRA
OVRA	OVTRA	OWRA	OWTRA	OXRA	OXTRA	OYRA	OYTRA	OZRA	OZTRA	POTRA
PTRA	QOTRA	QTRA	ROTRA	RTRA	SOTRA	STRA	TOTRA	TRA	TTRA	UOTRA
UTRA	VOTRA	VTRA	WOTRA	WTRA	XOTRA	XTRA	YOTRA	YTRA	ZOTRA	ZTRA

Se pide implementar el algoritmo y realizar las pruebas necesarias para verificar que funciona correctamente.

Parte 3:

Crea una **estructura lineal**, **Diccionario**, que permita almacenar una colección de términos. Ten en cuenta que para cada término, además queremos almacenar todas sus posibles variaciones obtenidas aplicando el método desarrollado en la parte 2. Para almacenar todas estas posibles variaciones se debe utilizar la clase ListaPalabras, desarrollada en la parte 1.

Los métodos a implementar son:

- void **insertarTermino**(String p): que recibe un objeto de tipo String y almacena el término en la colección. Este método además de añadir (en orden alfabético

ascendente) el término en el diccionario, deberá generar y almacenar todas sus posibles variaciones obtenidas aplicando el método explicado en la parte 2. Si el término palabra ya existe no deberá ser añadido a la colección y el método deberá mostrar un mensaje informando que ya existe dicho término en la colección.

- void **eliminarTermino(String p)**: recibe un objeto de tipo String y elimina dicho término de la colección. El borrado de un término también implica también el borrado de sus variaciones).
- int **buscarTermino**(p): que recibe un objeto de tipo String, busca el término en la colección **Diccionario**, mostrando sus posibles variaciones. Además, el método debe devolver la posición del término en el diccionario. Si no existe el término devuelve -1. **Aclaración:** el método no busca el término en la lista de las posibles variaciones.

Se pide:

1. Implementar la estructura Diccionario
2. Desarrollar una clase Test que te permita probar la nueva estructura y cada uno de sus métodos.

Parte 4:

Debido al gran número de términos que se pueden almacenar en un diccionario, se ha planteado mejorar la eficiencia de la estructura de datos para que las operaciones de inserción, borrado y consulta sean más eficientes. En concreto, el objetivo de esta fase es desarrollar una nueva estructura, **DiccionarioT**, que permita insertar, buscar y eliminar términos del diccionario con complejidad logarítmica. Por supuesto, los términos en esta nueva estructura también deben ser almacenados de forma alfabética (ascendente). Como en la parte 3, la estructura deberá almacenar los términos y sus variaciones obtenidas aplicando el método descrito en la parte 2. Implementa los métodos **insertarTermino**, **buscarTermino** y **eliminarTermino**. La descripción de los métodos es similar a la expuesta en la parte 3, con la salvedad que ahora el método **buscarTermino** no tendrá que devolver la posición del término, sino simplemente se limitará a mostrar el término y todas sus posibles variaciones.

Se pide

1. Implementar la nueva versión de la estructura DiccionarioT tal y como se indica en este apartado.
2. Implementar una clase Test que permita probar cada uno de los métodos de la estructura Diccionario.

Algunos comentarios:

1. Para el desarrollo del caso práctico se utilizará la herramienta Eclipse. En concreto, tendréis que crear un proyecto de tipo Java. Cada una de las partes del caso práctico será un paquete distinto del proyecto, por ejemplo: parte1, parte2, parte3 y parte4. Esto nos permitirá evaluar cada parte de la práctica de forma independiente. Es necesario considerar que el hecho de que las partes sean independientes, no supone que no se pueda reutilizar parte del trabajo realizado en los apartados anteriores, si así se requiere y considera el/la alumno/a.

2. Todos los métodos deben incluir mensajes informativos sobre el resultado final de la operación. Es decir, se deberá informar si la operación se ha realizado correctamente o en caso contrario, la causa por la que no se ha podido completar.

3. Para limitar la complejidad del caso práctico, vamos a suponer que las palabras sólo están formadas por las letras del abecedario. Es decir, las palabras no podrán contener: números, caracteres especiales, signos de puntuación, espacios en blanco, tildes o diéresis. Además, vamos a suponer que todas las palabras y sus variaciones se almacenan en minúscula.

4. El código debe incluir además de los comentarios del código, comentarios sobre las decisiones tomadas. A continuación se muestra un ejemplo de comentario que se deberá incluir en el código para explicar las decisiones tomadas:

```
/*** DECISIÓN ***
En los métodos de borrado, hemos decidido que el
método además devuelva la posición en la que  se
encontraba el elemento.*/
```

Entrega del caso práctico:

El caso práctico deberá ser implementado obligatoriamente por grupos de 2 personas. Bajo ningún concepto se permitirá grupos de más de 2 personas. En general, tampoco se permitirán grupos individuales, salvo que el número de estudiantes matriculados en el grupo docente sea impar. Si no dispones de compañero/a de prácticas, comunícaselo a tu profesor/a de laboratorio lo antes posible, para que te puedan asignar un compañer/a.

No se permitirá crear grupos de prácticas con miembros de distintos grupos docentes. Es decir, obligatoriamente ambo/as alumno/as deberán estar matriculados en el mismo grupo reducido.

La fecha límite de entrega será el **3 de mayo a las 9:00 am**. La entrega se realizará en aula global (cada profesor/a de práctica publicará una tarea que permita dicha entrega online en el aula global del grupo reducido). Solamente uno/a de lo/as dos alumno/as que forman parte del grupo deberá realizar la entrega online. La entrega deberá consistir en un fichero ZIP cuyo nombre debe seguir el siguiente formato: NIA1_NIA2.zip,

siendo NIA1, NIA2 los NIAS de lo/as alumno/as que forman el grupo. El fichero debe contener con un fichero zip con el proyecto Eclipse obtenido con la opción “Exportar” de Eclipse, y opcionalmente se podrá entregar una memoria con documentación sobre el proyecto si lo/as alumno/as lo consideran necesario.

Evaluación del caso práctico

Lo/as profesor/as citarán en una sesión de defensa del caso práctico a lo/as alumno/as, para que ésto/as acrediten haber adquirido los conocimientos exigidos en la realización del caso práctico. Cada miembro del grupo será evaluado de forma individual. Entre otros elementos, se valorará:

- El seguimiento de las convenciones de Java para el nombrado de símbolos, y el uso adecuado de identificadores.
- Comentar el código de forma completa (pero no excesiva).
- Que el código sea fácil de leer (guardar estilo, sangrado de código).
- La elección de las estructuras de datos apropiadas en la implementación.
- El desarrollo de algoritmos eficientes.
- Que el código esté bien estructurado y no contenga instrucciones innecesarias.