Práctica 1. Clase vector estático

Sesiones de prácticas: 1

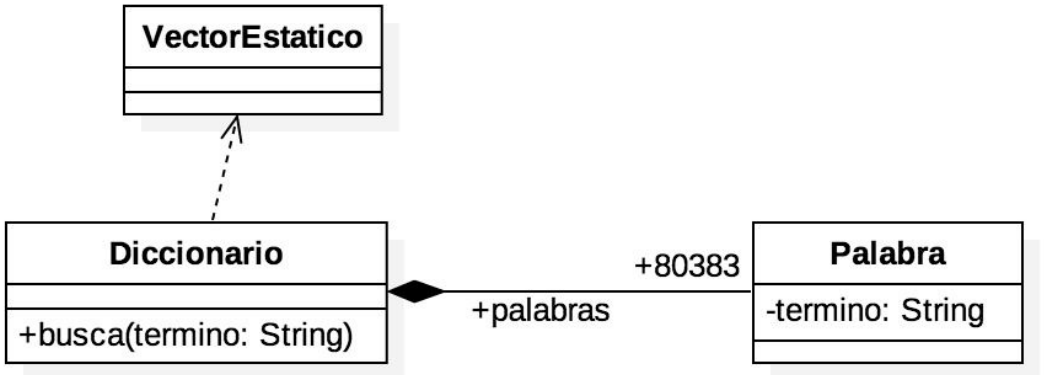
Objetivos

Creación de una clase vector estático y su instanciación a objetos de otra clase.

Descripción de la EEDD

Se quiere trabajar en un proyecto de texto predictivo. Para ello se va a empezar por almacenar un diccionario compuesto de 80.383 palabras del castellano.

Para ello se implementarán tres clases de acuerdo al siguiente UML:



- la clase Palabra que representa a una palabra
- la clase Diccionario que contiene a todo ese conjunto de palabras. Su constructor deberá tomar como parámetro el nombre del fichero, leerlo y rellenar el vector de tipo palabra.
- y la clase VectorEstatico que será el contenedor que usará Diccionario para almacenar dichas palabras. Deberá implementar las siguientes operaciones, aparte de los constructores.
    - *+insertar (p:Palabra, pos: long int);* que insertar una palabra en una posición
    - *+busquedaBin (p:Palabra): long int;* que realiza una búsqueda binaria y devuelve la posición donde se encuentra la palabra *p* en el vector, y -1 en caso de que no se encuentre.

Programa de prueba: Creación de un diccionario

El programa de prueba creará un diccionario a partir del fichero "listado_general.txt" . Este objeto contendrá el vector estático instanciado a objetos de tipo Palabra para mantener en memoria todo el conjunto de palabras de acuerdo al esquema anterior. El programa de prueba debe ser capaz de buscar y encontrar cualquier palabra válida en el diccionario mediante una búsqueda dicotómica, ya que el listado se da ordenado alfabéticamente.

NOTA1: Haced uso de excepciones en aquellos casos donde se considere conveniente.

NOTA2: en caso de tener problemas con la codificación del fichero, descargarlo directamente de:

https://github.com/javierarce/palabras

Repaso sobre lectura de ficheros de texto

Los flujos (*Streams*) son una abstracción habitual existente en lenguajes de programación orientada a objetos para el almacenamiento o acceso de información en archivos. Los flujos establecen una interfaz de uso homogénea que permite simplificar manipulación de archivos ocultando los detalles específicos de la comunicación con el dispositivo. En C++, la funcionalidad de un flujo de fichero está implementada en las clases std::fstream, std::ifstream y std::ofstream[1]. La primera se utiliza indistintamente para operaciones de lectura y escritura, mientras que las dos últimas son versiones simplificadas para cuando es necesario trabajar respectivamente con operaciones exclusivas de lectura o escritura sobre un fichero en particular.

Para acceder al contenido de un fichero en el sistema de ficheros, en primer lugar debe asociarse al objeto *stream* realizando un proceso que se conoce como apertura del fichero. De igual forma, al finalizar de realizar las operaciones de lectura y/o escritura hay que solicitar al sistema operativo que cierre el fichero para liberar los recursos asociados y evitar posibles pérdidas de información. Estas operaciones deben realizarse respectivamente con los métodos *open* y *close* de las clases citadas.

En C++, una vez abierto un fichero, pueden utilizarse diferentes funciones y métodos de las clases *stream* para recuperar o almacenar información. Muchas de estas funciones como *getline* o los operadores >> y << ya son conocidos puesto que se han utilizado con con los flujos *std::cin y std::cout*. A diferencia de los flujos asociados a ficheros, estos flujos se abren y cierran automáticamente.

Para finalizar, hay que tener en cuenta que las operaciones realizadas sobre ficheros no siempre pueden ser completadas, bien porque se produzca un error (error de lectura, acceso de lectura no permitido, …) o porque ocurran circunstancias específicas (se intenta leer al final de un fichero). Para comprobar estas situaciones, los flujos disponen también de diferentes métodos como *good*, para comprobar si la última operación realizada ha tenido éxito o *eof*, que indica si se ha intentado leer estando al final del fichero.

En el siguiente ejemplo se muestra un fragmento de código donde se resumen todos estos conceptos. Para más información, consultar la documentación de las clases:

[1] http://www.cplusplus.com/reference/istream/ifstream/, http://www.cplusplus.com/reference/ostream/ofstream/, http://www.cplusplus.com/reference/fstream/fstream/

```
std::ifstream fe; //Creamos un flujo de entrada, #include <ifstream>
std::string linea;
int total=0;

// Asociamos el flujo con un fichero y lo abrimos
fe.open( "listado-general.txt" );

if ( fe.good() ) {
    // Mientras no se haya llegado al final del fichero
    while( !fe.eof() ) {
        getline( fe, linea ); // Toma una línea del fichero
        if (linea!="") {      // Ignoramos líneas en blanco
            total++;
        }
    }
    std::cout << "Total de palabras en el archivo:" << total
              << std::endl;
    fe.close(); //Cerramos el flujo de entrada
} else {
    std::cerr << "No se puede abrir el fichero" << std::endl;
}
```

Estilo y requerimientos del código:

1. El código debe ser claro, tener un estilo definido y estar perfectamente indentado, para ello se pueden seguir algunos de los estilos preestablecidos para el lenguaje C++ (http://geosoft.no/development/cppstyle. html).

2. Deben comprobarse todas los posibles errores y situaciones de riesgo que puedan ocurrir (desbordamientos de memoria, parámetros con valores no válidos, etc.) y lanzar las excepciones correspondientes, siempre que tenga sentido. Leer el tutorial de excepciones disponible en el repositorio de la asignatura en docencia virtual.

3. Se valorará positivamente la calidad general del código: claridad, estilo, ausencia de redundancias, etc.