Práctica 4. Estructuras STL

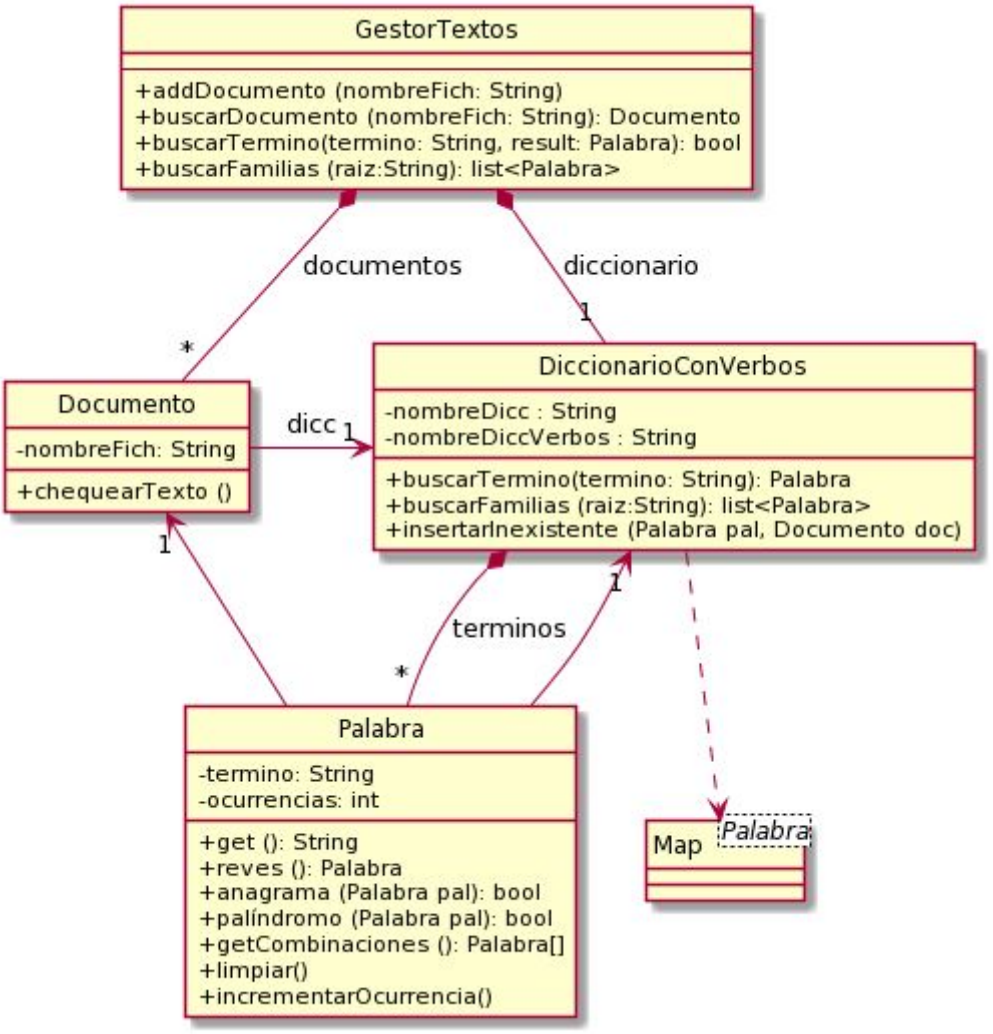
Sesiones de prácticas: 2

Objetivos

Aprender a utilizar las estructuras de datos recogidas en STL. Realizar un programa de prueba de las estructuras.

Descripción de la EEDD

La práctica reemplaza las estructuras de datos implementadas de forma nativa a contenedores de STL. El diseño también se actualiza según el siguiente esquema UML.



Descripción de la práctica:

La librería STL proporciona una serie de componentes que se pueden dividir en las siguientes categorías: algoritmos, contenedores, iteradores y funciones. La librería dispone de un conjunto prefabricado de las EEDD más comunes. Entre estas, se encuentran las implementadas en las prácticas anteriores que deben ser sustituidas por las que nos proporciona la librería STL.

La clase *GestorTextos* ahora puede realizar la búsqueda de una palabra o familia de palabras en el diccionario. Para buscar toda la familia de palabras dada una raíz se puede utilizar la función std::lower_bound. Además, el gestor de textos también contempla la incorporación de más diccionarios (castellano, inglés, francés, etc). Los documentos y diccionarios del gestor de textos deben almacenarse en un vector de STL.

En la clase *DiccionarioConVerbos* ahora se almacenan todos las palabras del diccionario y las conjugaciones de todos los verbos en una misma EEDD que será un Map<string,palabra>.

La clase *Documento* se sigue encargando del método *chequearTexto()*. Este método debe encargarse de meter en el diccionario todas las palabras que no se encuentran previamente (palabras inexistentes), así como de incrementar el atributo ocurrencias de la clase *Palabra* que se encarga de almacenar el

número de veces que aparece una palabra del diccionario en los textos analizados. Por defecto todas las palabras tienen que inicializar este valor a 0. Para incrementar este valor se utilizará el método *incrementarOcurrencias()* de la clase *Palabra*. Para evitar la inserción de valores repetidos, se debe comprobar antes que la palabra no se encuentra almacenada en la EEDD (hay que tener especial cuidado con las palabras que empiezan por mayúscula por ir detrás de un punto).

La clase *Palabra* ahora también se relaciona con la clase *Diccionario* y *Documento*. Cada palabra inicial debe mantener una referencia al diccionario en el que se encuentra. Las palabras inexistentes sólo tendrán la referencia al último documento en el que han sido encontradas.

Para los que trabajan en pareja, el método *chequearTexto()* también se encargará de evitar la inserción de nombres propios.

Programa de prueba

Crear un programa de prueba con las siguientes indicaciones:

- Implementar las clases anteriores de acuerdo al diagrama UML.
- Realizar el mismo proceso de prueba que en la Práctica 3, es decir, chequear el documento en busca de palabras inexistentes, pero teniendo en cuenta las modificaciones sobre el diagrama.
- Añadir un segundo documento, de los adjuntos en la Práctica 3, y chequear el documento.
- Mostrar por consola el número de veces que aparecen las palabras “mancha” y “estaban”.
- Buscar y mostrar por consola la familia de palabras de: flor, sal y mar.
- Mostrar por consola todas las palabras del diccionario que tengan asociado un documento (palabras inexistentes que han sido añadidas en el diccionario) junto con el nombre del documento en el que fueron halladas.
- Mostrar por consola el tamaño del mapa antes y después de chequear los documentos.

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.