Práctica 2. Implementación de una lista dinámica mediante plantillas y operadores en C++

Sesiones de prácticas: 2

Objetivos

Implementar y utilizar la clase `ListaDEnlazada<T>` y su clase auxiliar de tipo iterador ListaDEnlazada<T>::`Iterador` utilizando **patrones de clase y excepciones**. Programa de prueba para comprobar su correcto funcionamiento.

Descripción de la EEDD

Implementar la clase `ListaDEnlazada<T>` para que tenga toda la funcionalidad de una lista doblemente enlazada en memoria dinámica descrita en la Lección 7, utilizando patrones de clase y excepciones. Los métodos a implementar serán los siguientes:

- Constructor por defecto `ListaDEnlazada<T>()`
- Constructor copia `ListaDEnlazada<T>(const ListaDEnlazada<T>& origen)`.
- Operador de asignación (=)
- Obtener los elementos situados en los extremos de la lista: `T& inicio()` y `T& Fin()`
- Obtener un objeto iterador para iterar sobre una lista bidireccionalmente: `ListaDEnlazada<T>::Iterador iterador ()` e implementar la funcionalidad completa del iterador.
- Insertar por ambos extremos de la lista, `void insertaInicio(T& dato)` y `void insertaFin(T& dato)`
- Insertar un dato en la posición anterior apuntada por un iterador: `void inserta(Iterador &i, T &dato)`
- Borrar el elemento situado en cualquiera de los extremos de la lista, `void borraInicio()` y `void borraFinal()`
- Borrar el elemento referenciado por un iterador: `void borra(Iterador &i)`
- `tam(): entero`, que devuelve de forma eficiente el número de elementos de la lista
- `concatena(const ListaDEnlazada<T> &l):ListaDEnlazada<T>`, que devuelve una nueva lista con la concatenación de la lista actual (this) y la proporcionada por parámetro. Sobrecargar también el *operador+* para realizar la misma funcionalidad.
- El destructor correspondiente.

Descripción de la práctica: crear un gestor de imágenes

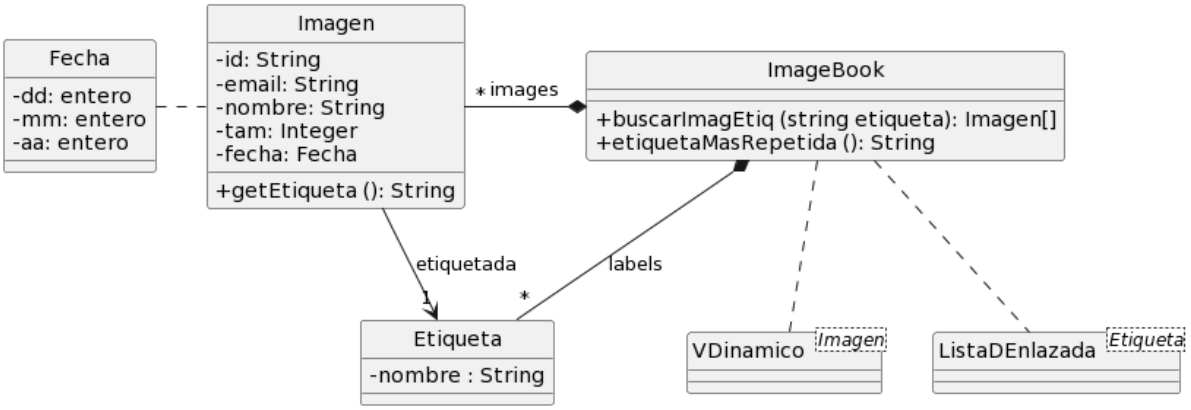
En la práctica anterior creamos un vector dinámico de imágenes. Como se visualiza en el UML anterior, en esta práctica el vector de imágenes se mantiene como el atributo *images* de la clase *ImageBook,* que funcionará como la clase gestora del conjunto de imágenes. A esta nueva clase también se añade un conjunto de etiquetas que pueden definir la temática de la imagen. Por ejemplo, la etiqueta "playa" en una foto del verano. Las etiquetas válidas para esta práctica ya estaban guardadas en el fichero "images_v1.csv", pero se pueden leer más fácilmente del fichero "etiquetas.txt" subido con la práctica. Por el momento, modificaremos la clase Imagen para que sólo tenga asociada una etiqueta (relación *Imagen::etiquetada*), que coincidirá con la primera de las etiquetas que aparece para cada imagen en el fichero "images_v1.csv". La relación *ImageBook::labels* se implementará mediante una lista doblemente enlazada `ListaDEnlazada<T>()`, tal y como indica el UML.

La funcionalidad de la clase *ImageBook* es la de gestionar el conjunto de imágenes y las etiquetas. La construcción de esta clase requiere la lectura de los dos ficheros adjuntos a la práctica: el de imágenes de la Práctica 1 ("images_v1.csv") y el de etiquetas ("etiquetas.txt"). Para implementar el constructor de ImageBook seguir lo siguientes pasos:

1. Leer el fichero *"etiquetas.txt"* y cargarlos en la lista asociada a *ImageBook:labels*.
2. Leer cada una de las imágenes del fichero *"images_v1.csv"* y para cada una de ellas:
    a. Crear un objeto Imagen con todos los atributos del fichero (menos etiqueta).
    b. Buscar en *ImageBook::labels* la primera etiqueta que aparece en el fichero y enlazar correctamente la asociación *Imagen::etiquetada*.
    c. La nueva imagen creada se añade ahora al vector del atributo *ImageBook::images.*

Además del constructor, la clase implementa la función *ImageBook::buscarImagEtiq (string etiqueta): Imagen[]* que localiza y devuelve todas aquellas imágenes que tengan la etiqueta dada como parámetro. El contenedor de salida será una lista (ListaDEnlazada<T>) (el "[]" en este ámbito simboliza "muchos") pero no deben ser objetos copia.

La función *ImageBook::etiquetaMasRepetida()* determina cual es la etiqueta más repetida en todas las imágenes almacenadas. En caso de empate devolver cualquier de ellas. La función *Image::getEquiqueta()* devuelve la cadena de caracteres asociada a la etiqueta.



Programa de prueba: probar el gestor de imágenes

1. Implementar la EEDD *ListaDEnlazada<T>* y el Iterador<T> con la funcionalidad señalada arriba y de acuerdo con la especificación de la Lección 7.
2. Instanciar la clase ImageBook según el diseño UML, rellenando la lista enlazada con las etiquetas del fichero *etiquetas.txt* y el vector dinámico de imágenes como en la práctica anterior (exceptuando que ahora la cadena con las etiquetas no forma parte de la clase Imagen). Para enlazar cada imagen con una etiqueta, durante la lectura del fichero, se obtiene la primera de las etiquetas de cada imagen, se busca dicha etiqueta en la lista de etiquetas (en *ImageBook::labels*) y se enlaza mediante la asociación *Imagen::etiquetada*.
3. Devolver cuál de las etiquetas es la más repetida utilizando la función ImageBook:etiquetaMasRepetida().
4. Devolver y mostrar por pantalla todas aquellas imágenes (id, usuario) con la etiqueta "playa" y posteriormente las que tengan la etiqueta "comida".
5. Unir ambas listas resultantes en una nueva lista resultado usando la función concatenar, comprobando que el resultado es idéntico usando el *operator+*.
6. **Para los que trabajan por parejas**: medir los tiempos de ejecución de la operación anterior e implementar la primitiva de la lista `ListaDEnlazada<T>::Iterador busca (T &dato)` que devuelve un iterador que referencie a un dato en la lista igual al suministrado[1]. Para probarla, modificar el código del apartado 2 para que utilice dicho método cada vez que sea necesario localizar una etiqueta para asociarla con su imagen[2].

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).

[1] Internamente utilizar el operador == para las comparaciones que, en el caso de las Imágenes consideraba el id de las mismas

[2] En este caso, dejar comentado el código original que realizaba la búsqueda sobre la lista iterando sobre sus componentes, que ahora se resolverá directamente llamando únicamente al nuevo método de forma adecuada. Ojo, hay que tener en cuenta que si una etiqueta no existe el método deberá devolver un iterador final (internamente con valor nullptr como nodo seleccionado)

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.