Práctica 4. Estructuras STL

Sesiones de prácticas: 2

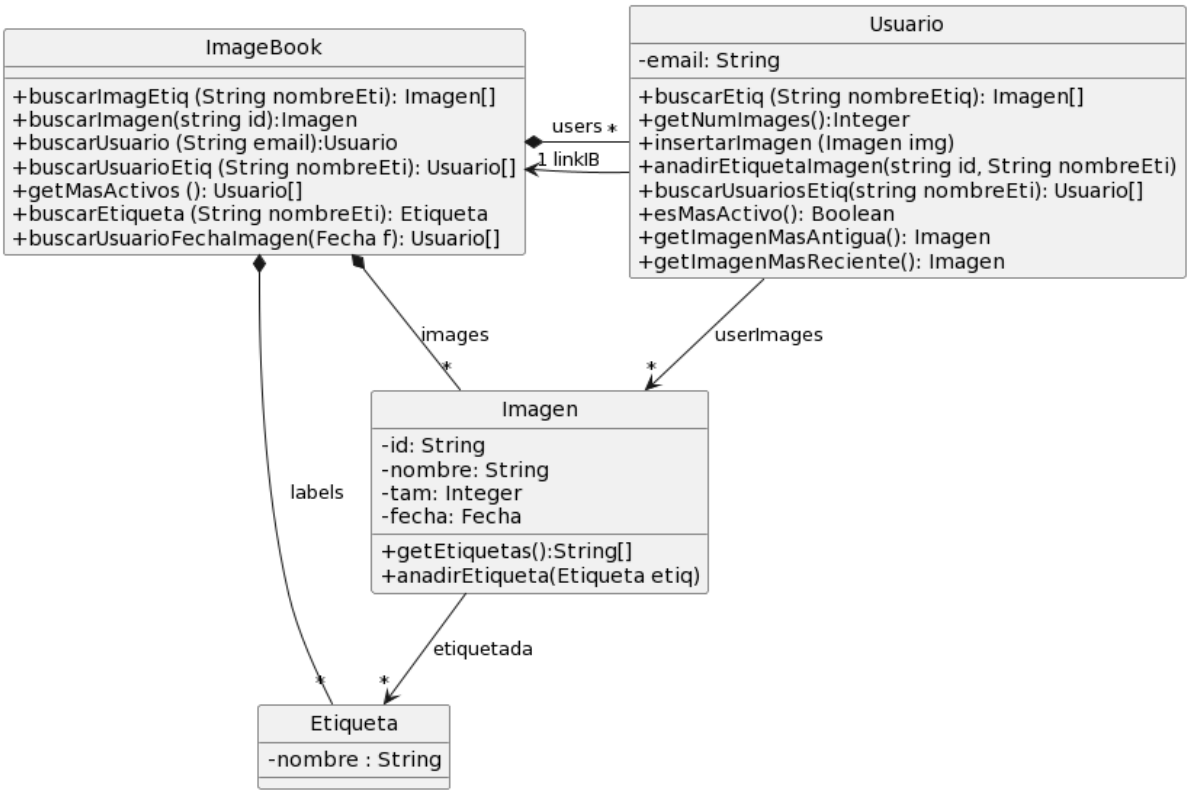
Importante: esta práctica y las siguientes deben ser realizadas por los estudiantes que convalidaron las tres primeras prácticas.

Objetivos

Aprender a utilizar las estructuras de datos (EEDD) recogidas en STL. Aprender a gestionar la inclusión circular de clases de objetos[1]. Realizar un programa de prueba de las estructuras.

Descripción de la EEDD

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



Descripción de la práctica:

[1] Cuando dos clases se referencian mutuamente deben utilizarse declaraciones adelantadas de tipos (*forward declaration*) para evitar problemas con la inclusión condicional de los ficheros cabecera https://es.cppreference.com/w/cpp/language/class

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 predefinido de las EEDD más comunes y, entre ellas, se encuentran las implementadas en las anteriores prácticas. En esta práctica vuestras EEDD deben ser sustituidas por las que nos proporciona la librería STL.

La clase ImageBook, al igual que en anteriores prácticas, se encargará de la gestión del conjunto de imágenes. Además, la clase Imagen tiene una asociación "a muchos" con la clase Etiqueta, por lo tanto, en esta práctica se almacenarán todas las etiquetas que tiene asociadas una imagen, no solamente la primera. También se ha incluido una nueva relación (*Usuario:linkIB*) entre las clases Usuario e ImageBook, de manera que cualquier usuario puede llevar a cabo acciones propias de ImageBook. Es importante tener en cuenta que entre las clases Usuario e ImageBook existe una inclusión circular que debe ser gestionada correctamente.

En cuanto a las EEDD, en la clase ImageBook los usuarios se almacenarán en un *Map<String, Usuario>* cuya clave vendrá determinada por el atributo *email* de la clase Usuario. Las imágenes, por su parte, se almacenan en un *Vector<Imagen>*. Por último, las etiquetas, se almacenan en una *List<Etiqueta>*. En cuanto a la clase Usuario, la asociación de una a muchos se representa con un *Map<String, Imagen*>* cuya clave vendrá determinada por el atributo *id* de la clase Imagen. Por último, la asociación de uno muchos de la clase Imagen con Etiqueta, vendrá representada por un *Deque<Etiqueta*>*. Resumiendo, los contenedores a usar para las cinco relaciones "a muchos" son:

- (*Map<String, Usuario>*) ImageBook::users
- (*List<Etiqueta>*) ImageBook:labels
- (*Vector<Imagen>*) ImageBook:images
- (*Map<String, Imagen*>*) Usuario::userImages
- (*Deque<Etiqueta*>*) Imagen::etiquetada

La nueva funcionalidad de las clases es la siguiente considerando que <u>los métodos nunca devuelven objetos copia</u>:

ImageBook:

- *Etiqueta* buscarEtiqueta(string nombre)* busca la etiqueta cuyo nombre se pasa por parámetro.
- *Usuario[] buscarUsuarioFechaImagen(Fecha fecha)* busca los usuarios que han publicado imágenes en una fecha determinada.

Imagen:

- *anadirEtiqueta(Etiqueta *etiqueta)* que añade una nueva etiqueta a la imagen.

Usuario:

- *Imagen* buscarImagen(string id)* busca una imagen del usuario a partir de su id.
- *anadirEtiquetaImagen(string id, string nombreEti)* añade una nueva etiqueta *(nombreEti)* a la imagen cuyo *id* se pasa por parámetro.
- *Usuario[] buscarUsuariosEtiq(string nombreEti)* busca a todos los usuarios con los que comparte una etiqueta.
- *bool esMasActivo()* comprueba si el usuario es el más activo de la red social.
- *Imagen* getImagenMasAntigua()* devuelve la imagen más antigua que ha publicado el usuario.
- *Imagen* getImagenMasReciente()* devuelve la imagen más reciente que ha publicado el usuario.

Programa de prueba

Crear un programa de prueba con las siguientes indicaciones:

- Reemplazar las EEDD implementadas en prácticas anteriores por los contenedores STL mencionados.
- Instanciar el mapa con objetos de tipo *Usuario* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo.
- Instanciar el vector con objetos de tipo *Imagen* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo. En esta práctica se almacenarán todas las etiquetas asociadas a la imagen, no solamente la primera como ocurría en la anterior.
- Instanciar la lista con objetos de tipo *Etiqueta* con el fichero adjunto. La clase *ImageBook* toma el nombre del fichero en el constructor para la lectura del mismo.
- El usuario noelia30@hotmail.com quiere incluir la etiqueta "playa" en una de sus imágenes cuyo id es 625722993.
- El usuario kenny_ohara73@yahoo.com quiere modificar la última imagen que ha subido. Encontrar dicha imagen y añadirle la etiqueta "viernes".
- El usuario elton.botsford@yahoo.com quiere conocer a todos los usuarios con los que comparte la etiqueta "arroz". Mostrar el número de usuarios obtenido y su email.
- Buscar a los usuarios que publicaron una imagen el día 7/9/2021 y mostrar sus datos. De entre todos los usuarios, mostrar quién ha publicado más imágenes.
- Comprobar si el usuario chesley.gerlach@hotmail.com es el más activo de la red social.

Nota: Para los que trabajan en parejas:

- Implementar el método *ImagenBook::Usuario[] buscarUsuariosPremium()* que busca el o los usuarios más antiguos de la red social (aquellos que fueron los primeros en publicar una imagen) y mostrar sus datos.

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.