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

Sesiones de prácticas: 2

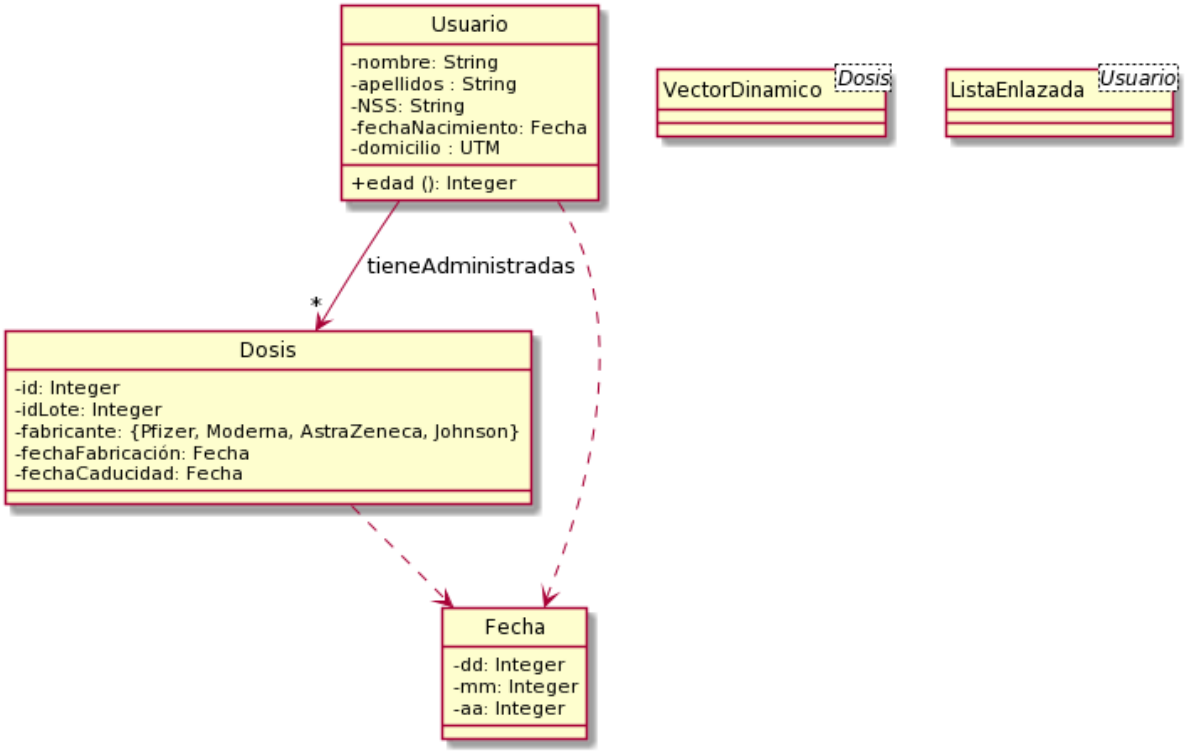
Objetivos

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

Descripción de la EEDD

Implementar la clase `ListaEnlazada<T>` para que tenga toda la funcionalidad de una lista enlazada en memoria dinámica descrita en la Lección 6, utilizando patrones de clase y excepciones. Los métodos a implementar serán los siguientes:

- Constructor por defecto `ListaEnlazada<T>`
- Constructor copia `ListaEnlazada<T>(const ListaEnlazada<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: `ListaEnlazada<T>::Iterador<T> iterador()`. Se debe implementar la clase `Iterador<T>` completa tal y como está en la lección 6.
- 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<T> &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<T> &i)`
- `int tam()`, devuelve de forma eficiente el número de elementos de la lista
- El destructor correspondiente.
- El método `bool buscar(T &dato, Iterador<T> &it)`, que busca el *dato* de tipo *T* en la lista y devuelve *true* en caso de ser encontrado y el iterador a la posición localizada. En caso de no encontrarse, devuelve *false*.



Programa de prueba: crear un gestor de textos

Las dosis de la práctica anterior van a ser suministradas a los usuarios. El diagrama UML anterior describe la administración de dosis a los usuarios. Por el momento sólo nos indica que a los usuarios se les administra una sola dosis.

La clase **Usuario** tiene una serie de atributos que la definen, entre ellos, el *número de la seguridad social* que es una cadena, la *fecha* de nacimiento que es de tipo *Fecha* y el domicilio que viene dado por sus coordenadas *UTM* (en esta práctica no usaremos las coordenadas UTM, se usarán en la práctica 6, pero se incluye ya para no tener que modificar el fichero de usuarios que os damos más adelante). El método *edad():Integer* se calcula a partir de la fecha de nacimiento del usuario. Los usuarios se almacenan en una lista simplemente enlazada. Por tanto, la clase *ListaEnlazada<T>* se instanciará con los usuarios *(ListaEnlazada<Usuario>)*

Las dosis se almacenan en un vector dinámico como en la Práctica 1, pero en esta ocasión debe estar ordenado por la fecha de fabricación. Las dosis se administran a los usuarios por su fecha de fabricación, por tanto, el primer usuario de la lista recibirá la primera dosis del vector ordenado que será la primera que se fabricó. (Nota: hay dosis que por su fecha de fabricación aún no se han fabricado, no tiene relevancia para la práctica).

Para probar esta práctica:

- Implementar la EEDD *ListaEnlazada<T>* con la funcionalidad señalada arriba y de acuerdo con la especificación de la Lección 6.

- Instanciar la lista enlazada con los usuarios usando el fichero *usuarios.txt*
- Instanciar dosis como un vector de dosis, similar a como se hizo en la Práctica 1 usando el fichero *dosis.txt*
- Listar por pantalla las dosis administradas a los 50 primeros usuarios de la lista.
- Mostrar por pantalla la información de los usuarios con NSS 1491983009, 1280280451, 1696453083, incluyendo su edad y la dosis administrada.
- Buscar de dicha lista aquellos usuarios que tienen como nombre Joan, indicar cuántos hay y borrarlos de la lista. Volver a buscar Joan en la lista de usuarios para comprobar que se han borrado correctamente.
- Mostrar el tamaño de la lista de usuarios una vez realizado el apartado anterior.
- **Para los que trabajan por parejas**: buscar cual es el nombre que más se repite en la lista de usuarios y construir una nueva lista con dichos usuarios. Mostrar la nueva lista y su tamaño. Este apartado debe hacerse antes del apartado 6 (borrar los usuarios con nombre Joan).

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.