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

Sesiones de prácticas: 2

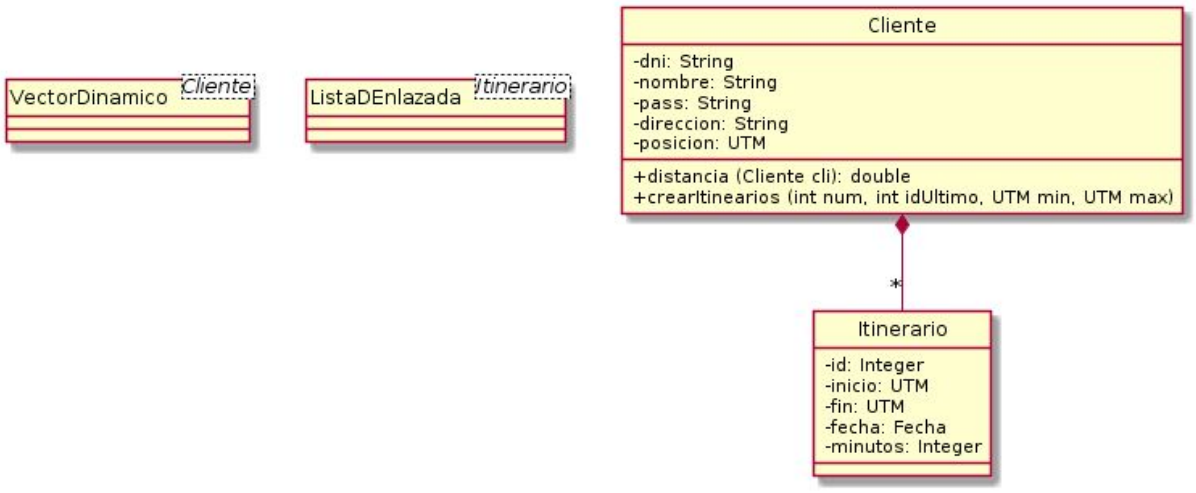
Objetivos

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: `ListaDEnlazada<T>::Iterador 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 y la proporcionada por parámetro.
- El destructor correspondiente.



Programa de prueba: creación de una lista itinerarios asociados a los clientes

La empresa de alquiler de motos por tiempo llamada EcoCityMoto se ha creado para facilitar el transporte urbano mediante vehículos eléctricos. Las motos estarán aparcadas por la ciudad y cualquiera que esté dado de alta en la aplicación puede reservar una que esté cerca de su ubicación y utilizarla. El cobro se realiza por el número de minutos que ha utilizado el vehículo.

Ahora mismo está cambiando la aplicación y tiene una serie de clientes ubicados en un fichero que han insertado en un vector (Práctica 1). Todos estos clientes ya han realizado al menos un itinerario con un vehículo de la empresa. Esta información sobre itinerarios se guardará en una lista doblemente enlazada. El esquema general de funcionamiento lo observamos en el UML de la figura.

Para probar esta práctica:

- Partir de la Práctica 1 con el conjunto de clientes guardados en memoria.
- Implementar la EEDD ListaDEnlazada<T> de acuerdo con la especificación de la Lección 7.
- La función Cliente::*crearItinerarios()* crea una lista de *num* itinerarios de forma aleatoria. Para ello dará un Id único a cada itinerario a partir del último asignado *IdUltimo* y añadirá una fecha aleatoria a partir de enero de 2019, además de un número de minutos aleatorio. Las posiciones de inicio y fin también serán aleatorias. Para ser coherentes con las posiciones de los clientes, averiguad previamente las coordenadas máximas (máxima lat y long) y mínimas (mínimas lat y long) para que se generen posiciones en el mismo rango.
- Los itinerarios generados deben guardarse en la lista doblemente enlazada asociados a cada Cliente como se indica en el UML

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.