# 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 de iteración, ListaEnlazada<T>::`Iterador` utilizando **patrones de clase y excepciones**. Integración de esta clase en un diseño de clases y 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 simplemente 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 sin modificar la lista: `T& inicio()` y `T& Fin()`
- Obtener un objeto iterador para iterar sobre la lista: `ListaEnlazada<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 en O(n) un dato en la posición anterior apuntada por un iterador: `void inserta(Iterador &i, T &dato).`
- Insertar en O(1) un dato en la posición siguiente apuntada por un iterador: `void insertaDetras(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 ListaEnlazada<T> &l):ListaEnlazada<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.

**Proyecto de prueba: Gestión de aeropuertos**

VuelaFlight es un nuevo proyecto para gestionar todos los aeropuertos del mundo. En esta práctica trabajaremos con el vector dinámico de aeropuertos de la Práctica 1, y además se instanciará una lista simplemente enlazada de rutas, que conectarán un aeropuerto origen con otro de destino con vuelos de una determinada aerolínea identificada por su código IATA[1].

El siguiente UML representa el diseño de *VuelaFlight* como la clase gestora de aeropuertos y rutas. En esta práctica utilizaremos una nueva versión del archivo de Aeropuertos, aeropuertos_v2.csv, donde su segundo campo se corresponde ahora con su código IATA para poder relacionarlo con la información de las rutas. Se mantiene el vector dinámico de objetos de tipo *Aeropuerto*, ahora como composición de la clase gestora *VuelaFlight.* A esta nueva clase se añade otra composición con la clase *Ruta*, considerada como trayecto válido entre un aeropuerto de origen y otro destino. Estas rutas se encuentran en el fichero adjunto "rutas_v1.csv". En este fichero, la primera columna representa a las siglas IATA de la compañía aérea y las dos siguientes representan a los aeropuertos origen y destino por este orden. Las siglas de cada aeropuerto siguen el código IATA, que identifica con 3 siglas cualquier aeropuerto del mundo, p.e. GRX es el aeropuerto Granada-Jaén, AGP el de Málaga, MAD corresponde al de Madrid-Barajas, etc. Como se observa en el UML, un objeto de tipo *Ruta* debe enlazar con dos de tipo *Aeropuerto*. Para inicializar los datos de dichas clases se deben seguir los siguientes pasos:

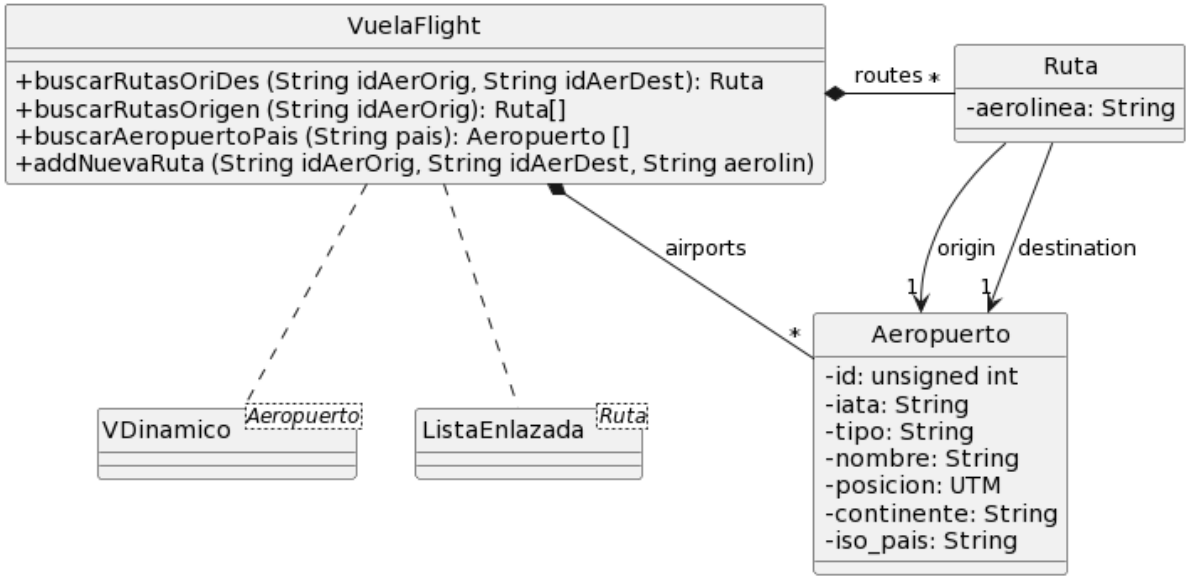
1. Leer el fichero *"aeropuertos_v2.csv"* y cargarlos en el vector dinámico *VuelaFlight:airports*. Ordenar el vector de forma ascendente usando el identificador *IATA*. Para ello será necesario cambiar el código del operator< (puede que algún operador más) en la clase Aeropuerto.
2. Leer el fichero "rutas_v1.csv" y añadir sus datos de la misma forma a la lista simplemente enlazada definida como *VuelaFlight:routes*. El identificador de la aerolínea será el valor alfanumérico de la primera columna. Por ejemplo, "IBE" identifica a Iberia.
3. Para enlazar cada ruta con los dos aeropuertos de origen y destino:
    a. Para cada ruta de *VuelaFlight::routes*,
        i. buscar en tiempo logarítmico en *VuelaFlight:airports* el identificador IATA del aeropuerto de origen, obtener el objeto correspondiente y enlazarlo en *Ruta::origin*
        ii. hacer lo mismo con el aeropuerto de destino para enlazarlo el aeropuerto destino en *Ruta::destination*

Por el momento, la funcionalidad de la clase *VuelaFlight* es básica, se va a encargar de hacer algunas búsquedas:

[1] International Civil Aviation Organization

- *buscarRutasOriDes()*, que devuelve la ruta (si la hay) entre un aeropuerto de origen y otro de destino. Pudiera ocurrir que más de una compañía hiciera la misma ruta, pero bastará con encontrar una de ellas.
- *buscarRutasOrigen ()*, que devuelve una lista enlazada con todas las rutas que salen de un aeropuerto origen.
- *buscarAeropuertoPais (),* obtiene un vector con todos los aeropuertos que existen en un país dado.
- *addNuevaRuta()*, que crea una nueva ruta entre dos aeropuertos origen y destino.

**IMPORTANTE: no devolváis objetos copia**




**Programa de prueba I: probar la lista enlazada con enteros**

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

Probar la robustez de la lista implementando la siguiente funcionalidad:

- Crear una lista de enteros inicialmente vacía.
- Insertar al final de la lista los valores crecientes desde 101 a 200.
- Insertar por el comienzo de la lista los valores decrecientes desde 98 a 1
- Insertar el dato 100 delante del 101
- Insertar el dato 99 detrás del 98.
- Mostrar la lista resultante por pantalla.
- Borrar de la lista los 10 primeros y los 10 últimos datos.
- Borrar de la lista todos los múltiplos de 10
- Mostrar la lista resultante por pantalla.

## Programa de prueba II: probar la funcionalidad de VuelaFlight

1. Instanciar la clase *VuelaFlight* según el diseño UML y rellenar el vector y la lista como se ha descrito anteriormente.
2. Buscar si hay ruta entre el aeropuerto de Barcelona (BCN) con el de Estambul (IST).
3. Buscar también si hay ruta entre el aeropuerto de Granada-Jaén (GRX) con algún aeropuerto inglés (GB).
4. Añadir en O(1) una nueva ruta entre el aeropuerto de Granada-Jaén (GRX) con el de París Charles de Gaulle (CDG) de la compañía Iberia (IBE). Las rutas deben ir y volver entre ambos aeropuertos.
5. **Para los que trabajan por parejas**: Buscar todas las rutas existentes entre España y Portugal. Medir los tiempos asociados.

## 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.