# Práctica 4. Estructuras STL

## Sesiones de prácticas: 2

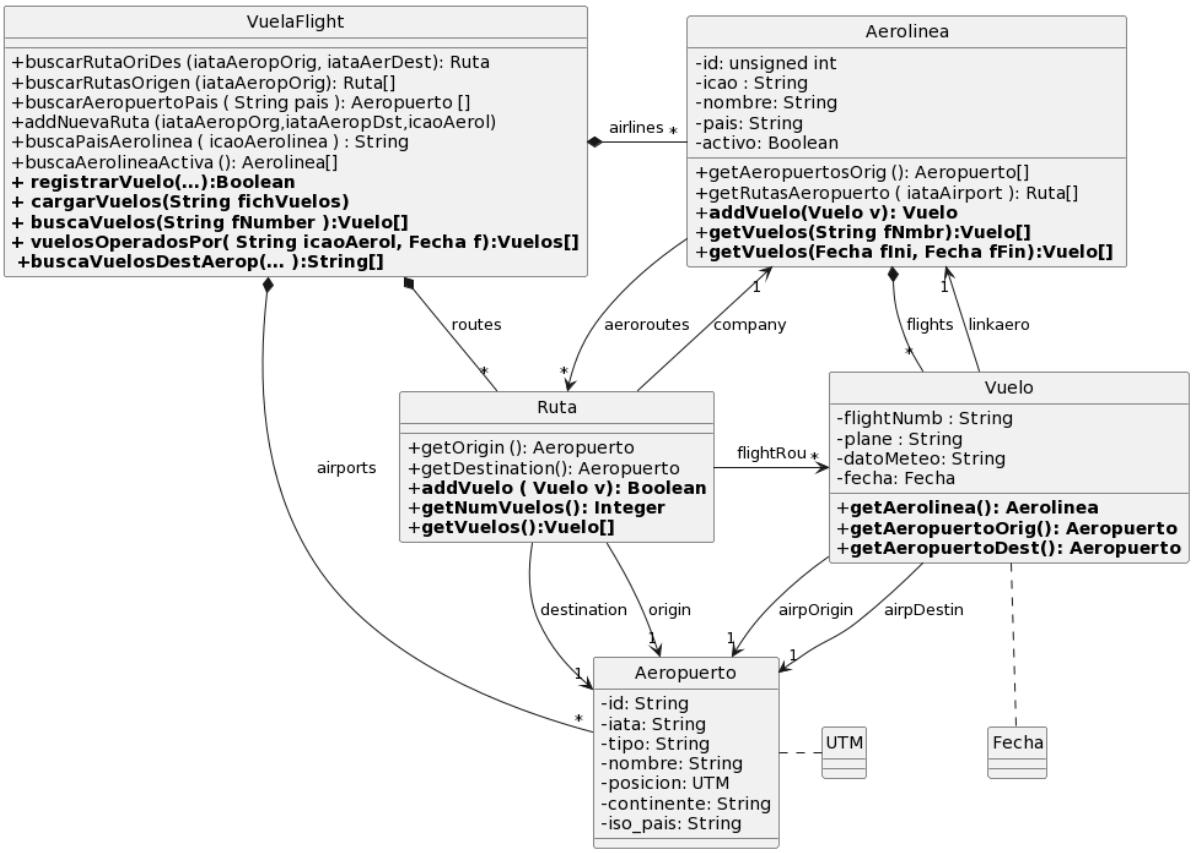
*Importante: esta práctica y las siguientes deben ser realizadas por los estudiantes que convalidaron las tres primeras prácticas.*

### Objetivos

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

### Descripción de las EEDD

En esta práctica reemplazaremos las EEDD implementadas en sesiones anteriores por contenedores de STL. El diseño también se actualiza según el siguiente esquema UML para añadir la **clase Vuelo**:




### Descripción de la práctica

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.

En esta práctica añadiremos la **clase Vuelo** que representa el registro de un vuelo operado por una aerolínea entre dos Aeropuertos. Cada Vuelo dispondrá en el sistema de la información asociada de la aerolínea que lo opera (*Vuelo::linkaero*) que se identifica a partir de los tres primeros caracteres del **número de vuelo**[1], los aeropuertos de origen, destino, la fecha de realización y las condiciones meteorológicas durante el trayecto. Los registros de Vuelos se almacenan en su respectiva Aerolínea (relación *Aerolinea::flights*). Además, los vuelos comerciales, es decir, aquellos que la compañía opera en alguna de sus rutas registradas (*Aerolínea::aerorutes*) también se añadirán a dicha ruta (*Ruta::flightRou*). Para **cargar los vuelos** en el sistema, disponibles en el fichero *vuelos_v1.csv*, y establecer sus relaciones adecuadas, utilizaremos el método *VuelaFlight::cargarVuelos()* descrito más abajo una vez se hayan cargado el resto de datos según se hizo en las prácticas anteriores. Es importante tener en cuenta que entre las clases Aerolínea y Vuelo existe una **inclusión circular** que debe ser gestionada correctamente[2]. La clase **Fecha** no hay que implementarla, se facilitará para que podáis usarla directamente.

En cuanto a las EEDD, en la clase **VuelaFlight** los aeropuertos se almacenarán en un *vector<Aeropuerto>* ordenados por su código IATA. Las Aerolíneas se almacenarán en un *map<string, Aerolinea>* usando como clave el ICAO de cada aerolínea. Las Rutas, por su parte, se almacenan en una *list<Ruta>*. En la clase **Aerolínea**, las rutas comerciales que opera se almacenan en un *deque<Ruta*>*. Como **los identificadores de un vuelo se pueden repetir** para diferentes trayectos de una misma ruta realizados en uno o varios días, estos se almacenarán en un *multimap<string, Vuelo>* donde la clave utilizada será el propio identificador de cada vuelo. En la clase **Ruta** se almacenarán también las referencias de aquellos vuelos que pertenezcan a dicha ruta mediante un *list<Vuelo*>*.

Resumiendo, los contenedores a usar para las seis relaciones "uno a muchos" son:

- Vuelaflight::airports→ *std::vector<Aeropuerto>*
- Vuelaflight::routes→ *std::list<Ruta>*
- Vuelaflight::airlines→ *std::map<string, Aerolinea>* con clave icao de la aerolínea
- Aerolinea::flights→ *std::multimap<string,Vuelo>* con clave el identificador de vuelo
- Aerolinea::aeroroutes→*std::deque<Ruta*>*
- Ruta::flightRoute→ *std::lista<Vuelo*>*

La nueva funcionalidad de las clases es la siguiente considerando que los métodos nunca devuelven objetos copia de los objetos gestionados por el sistema:

**Vuelo**

- `getAerolinea():Aerolinea`
- `getAeropuertoOrigen/Destino():Aeropuerto`

**Aerolinea**

---

[1] Ejemplo: el vuelo AEA5201 está operado por AirEuropa, AEA

[2] 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

- `addVuelo(Vuelo v):Vuelo`, añade a la aerolínea una copia de un vuelo, previamente inicializado con su aerolínea y aeropuertos relacionados. Si el vuelo no está correctamente inicializado o la aerolínea no se corresponde devolverá *nullptr*. En otro caso, lo añadirá y devolverá su nueva dirección de memoria. Además, una vez añadido, si el vuelo pertenece a alguna ruta de la propia aerolínea se añadirá también a la misma (*Ruta::addVuelo*). Este método se utilizará en *VuelaFlight::registrarVuelo()*.
- `getVuelos( String fNumber ):Vuelo[]`, Devuelve los vuelos registrados con el identificador indicado. Puesto que puede haber varios vuelos con el mismo identificador, utilizar el método *multimap::find* que encuentra el primer dato y para encontrar el resto, hay que iterar sobre todos los vuelos con dicho identificador.
- `getVuelos( Fecha fIni , Fecha fFin ):Vuelo[]`, devuelve los vuelos registrados en un rango de fechas determinado.

**Ruta**

- `addVuelo(Vuelo v):Boolean`, añade el registro de un vuelo de una aerolínea a la ruta. Debe verificar que la aerolínea y los aeropuertos de origen y destino coinciden con los de la ruta. Si el vuelo se consigue añadir, devuelve verdadero, o falso en el caso contrario.
- `getNumVuelos():Integer`, número de vuelos realizados en la ruta
- `getVuelos(): Vuelo[]`, devuelve los vuelos realizados en la ruta

**VuelaFlight**

- `registrarVuelo(String fNumber, String iataAeroOrig, String iataAeroDest, String plane, String datosMeteo, Fecha f):Boolean`, construye y añade un nuevo vuelo, debidamente inicializado, a la aerolínea en cuestión (usando *Aerolinea::addVuelo*). Devuelve verdadero si el vuelo se añade correctamente o falso si algún dato no fuera válido. Este método se utilizará en *cargarVuelos*.
- `cargarVuelos(String fichVuelos)`, método privado para cargar el fichero de vuelos. Llamar desde el constructor de la clase cuando ya se hayan cargado todos los datos previos según se realizó en las anteriores prácticas. Utilizar el método anterior para añadir cada vuelo leído.
- `buscaVuelos( String fNumber ):Vuelo[]`, devuelve los datos de los vuelos con identificador especificado.
- `vuelosOperadosPor(String icaoAerolinea, Fecha f):Vuelos[]`, información de vuelos operados por una aerolínea en una fecha indicada. *Nota: utilizar Aerolinea::getVuelos(fechaini,fechafin)*
- `buscaVuelosDestAerop(String paisOrig, String iataAeroDest):String[]`, busca identificadores de vuelo (únicos) desde cualquier aeropuerto de un país al aeropuerto especificado. *Nota: utilizar adecuadamente los métodos de VuelaFlight para obtener las rutas de los aeropuertos del país y, para cada ruta hacia el aeropuerto destino, obtener sus vuelos y usar un set para almacenar sus identificadores de vuelo de forma que se eliminen los duplicados.*

**IMPORTANTE: no devolváis copias de objetos que formen parte de composiciones**

## Programa de prueba

Crear un programa de prueba con las siguientes indicaciones:

1. Además de las funcionalidades de la Práctica 3 sobre VuelaFlight adaptadas convenientemente para trabajar con EEDD de STL, realizar también las siguientes operaciones:
2. Para los vuelos con identificador de vuelo AEA5201 y VLG2021, si existen, mostrar
    - Datos de la aerolínea que los opera ( nombre completo y país al que pertenece) e iata de los aeropuertos de origen, destino
    - Listado con todas las fechas y estado del tiempo en las que se ha efectuado en condiciones de lluvia o chubascos
3. Mostrar los modelos de aviones (únicos) utilizados en vuelos operados por Vueling, VLG, el 13/4/2018
4. Mostrar identificadores de vuelo (únicos) con destino a Londres (LHR, STN, LTN,LGW) desde cualquier aeropuerto español

### Para los que trabajan en parejas

- `VuelaFLight::buscaAeropuertosAerolinea(icaoAerolinea):Aeropuertos[]`, implementar este método que busca aeropuertos (sin repetir) donde opera una aerolínea específica, es decir, donde hay registrados vuelos con origen o destino operados por dicha compañía
- Probar el método al final del programa de prueba, mostrando ciudades donde se operan vuelos de Iberojet, EVE

### 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.