Práctica 4. Estructuras STL

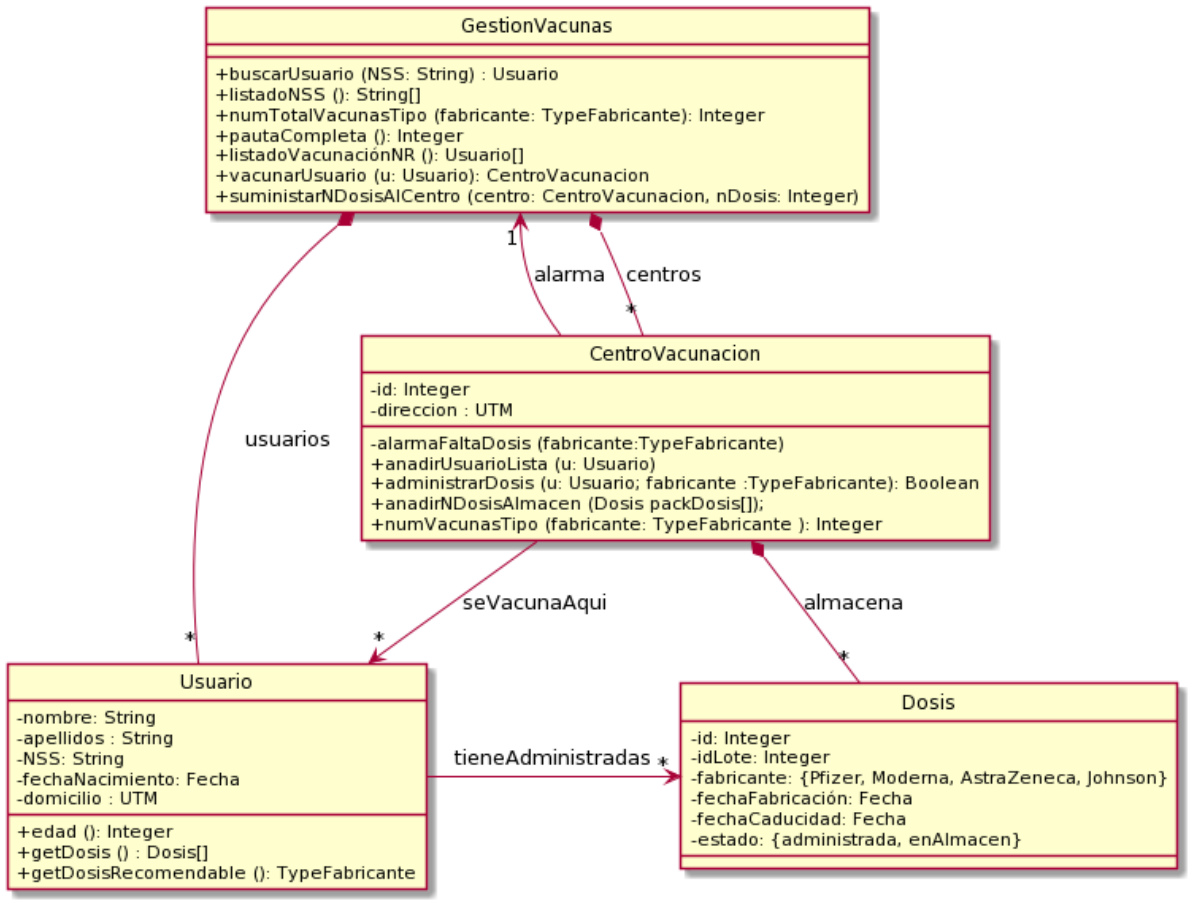
Sesiones de prácticas: 2

Objetivos

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

Descripción de la EEDD

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



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 prefabricado de las EEDD más comunes. Entre estas, se encuentran las implementadas en las prácticas anteriores que deben ser sustituidas por las que nos proporciona la librería STL.

La clase *GestionVacunas* se encargará de las vacunaciones de los usuarios a través del método *GestionVacunas::vacunarUsuario()*. Cuando un usuario acude a solicitar una vacuna al sistema, primero se localiza el centro de vacunación más cercano al usuario[1], después se asigna al usuario y posteriormente se devuelve en el método. El usuario sólo será vacunado si está registrado en un centro de vacunación. Posteriormente, se llamará al método *CentroVacunación::administrarDosis()* donde se indica el tipo de dosis que se va a administrar al usuario. Si el usuario se encuentra registrado en el centro de vacunación correspondiente, esto es, que en el CentroVacunacion se haya añadido al Usuario (asociación *seVacunaAqui*), entonces se procederá a su vacunación. Tras realizar dicha vacunación, se eliminará la vinculación del usuario con el centro para que no pueda vacunarse sin permiso, y se devolverá true. En el caso de que el usuario no se encuentre registrado en el centro de vacunación, se devolverá false. Si el centro de vacunación se quedara sin dosis lo notificará a *GestionVacunas* mediante el método *CentroVacunacion::alarmaFaltaDosis()*. Este método llamará a *GestionVacunas::suministarNDosisCentro()* que a su vez envía las 100 dosis libres siguientes al centro de vacunación correspondiente a través del método *GestionVacunas::anadirNDosisAlmacen()* que se usaran para vacunar a los usuarios de forma inmediata. Esta función *GestionVacunas::suministarNDosisCentro()* también puede usarse para inicializar las dosis en el sistema. Se podrá comprobar el número total de dosis de cada tipo almacenadas en los centros de vacunación mediante el método *GestionVacunas::numTotalVacunasTipo()*.

La clase *CentroVacunacion* representa los lugares donde los usuarios serán vacunados. Tiene una posición en el mapa dada por su posición UTM. **Cada centro tiene asociado una lista de usuarios registrados que serán los únicos que podrán ser vacunados en ese centro**. Los usuarios son registrados en el centro a través del método *CentroVacunacion::anadirUsuarioLista()*. Además, se podrá comprobar el número de dosis de cada tipo que quedan disponibles en el almacén de cada centro mediante el método *CentroVacunacion::numVacunasTipo()*.

En cuanto a las estructuras de datos, en la clase *GestionVacunas* los usuarios se almacenarán en un *Map<String, Usuario>* cuya clave vendrá determinada por el NSS del usuario. Los centros se almacenarán en un *Vector<Centro>*. En la clase *CentroVacunacion* los usuarios registrados se almacenarán en una *List<Usuario*>* y las dosis en un *Multimap<String, Dosis>* cuyas claves serán los fabricantes de las dosis. En la clase *Usuario*, las dosis administradas serán almacenadas en un *Vector<Dosis*>*. Obsérvese que las asociaciones son forzosamente definidas con puntero y las composiciones no tienen porqué, resultando más cómodo hacerlas sin puntero. El parámetro *packDosis* de la función CentroVacunacion::añandirNDosisAlmacen() puede implementarse como un vector.

---

[1] La distancia entre dos coordenadas UTM (un struct con dos campos de tipo float, latitud y longitud) se calcula como la distancia entre dos puntos https://es.wikipedia.org/wiki/Distancia

Programa de prueba

Crear un programa de prueba con las siguientes indicaciones:

- Implementar las clases anteriores de acuerdo al diagrama UML.
- Reemplazar las estructuras de datos implementadas en prácticas anteriores por los contenedores STL mencionados anteriormente.
- Instanciar el mapa con objetos de tipo *Usuario* con el fichero adjunto. La clase *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior.
- Instanciar el vector con objetos de tipo *CentroVacunacion* con el fichero adjunto. La clase *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior.
- Instanciar el vector con objetos de tipo *Dosis* de cada centro tomando el orden de lectura del fichero de dosis adjunto. *GestionVacunas* toma el nombre del fichero en el constructor como en la práctica anterior. Al primer centro de vacunación se le asignarán las primeras 8.000 dosis, al segundo las siguientes 8.200, al tercero las siguientes 8.500, al cuarto las siguientes 5000 y al último las 50 siguientes. Tened en cuenta que quedan 250 dosis sin repartir que se emplearán para proveer a los centros que se queden sin dosis.
- Mostrar el número de dosis total almacenadas de cada tipo usando el método *GestionVacunas::numTotalVacunasTipo()*.
- Vacunar a todos los usuarios cuyo NSS sea par empleando para ello el *vecAuxiliar* de la práctica anterior y buscar dichas claves en el *Map<String, Usuario>*.
- Vacunar a todos los usuarios con más de 65 años.
- Vacunar a todas las usuarias que se llaman Eva, mostrar su nombre completo y el id y la dirección del centro donde han sido vacunadas.
- Si en algún momento del proceso de vacunación un centro se queda sin dosis en almacén, leer del fichero las primeras 100 dosis sobrantes (a partir de la última dosis leída del fichero), añadirlas al multimap y proseguir con la vacunación.
- De igual forma que en la práctica anterior, mientras haya dosis de las recomendadas se ponen éstas, si no se ponen de cualquier otro fabricante. **Siempre se da prioridad a las dosis que ya están en el CentroVacunación**, por lo que si en un centro no quedan dosis de un tipo determinado, se vacunará con una dosis de otro tipo, independientemente de si hay del tipo recomendado en las dosis no leídas del fichero.
- Después de la vacunación, mostrar de nuevo el número total de dosis almacenadas de cada tipo.

Nota: Para los que trabajan en parejas:

- Completar la pauta de vacunación de los usuarios. Para ello, crear la función *Usuario::dosisPorAdministrar(): Integer* que devuelve el número de dosis que le quedan al usuario según su edad para completar la pauta. Recordad que los menores de 75 años requieren dos dosis (excepto aquellos entre 0 y 12 años, que no se vacunan) y los mayor de 75 tres.
- Mostrar a todos los usuarios a los que se les ha administrado la pauta completa con las dosis recomendadas.

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.