Práctica 4. Estructuras STL

Sesiones de prácticas: 2

Objetivos

Aprender a utilizar las estructuras de datos recogidas en STL. Realizar un programa de prueba de las estructuras.

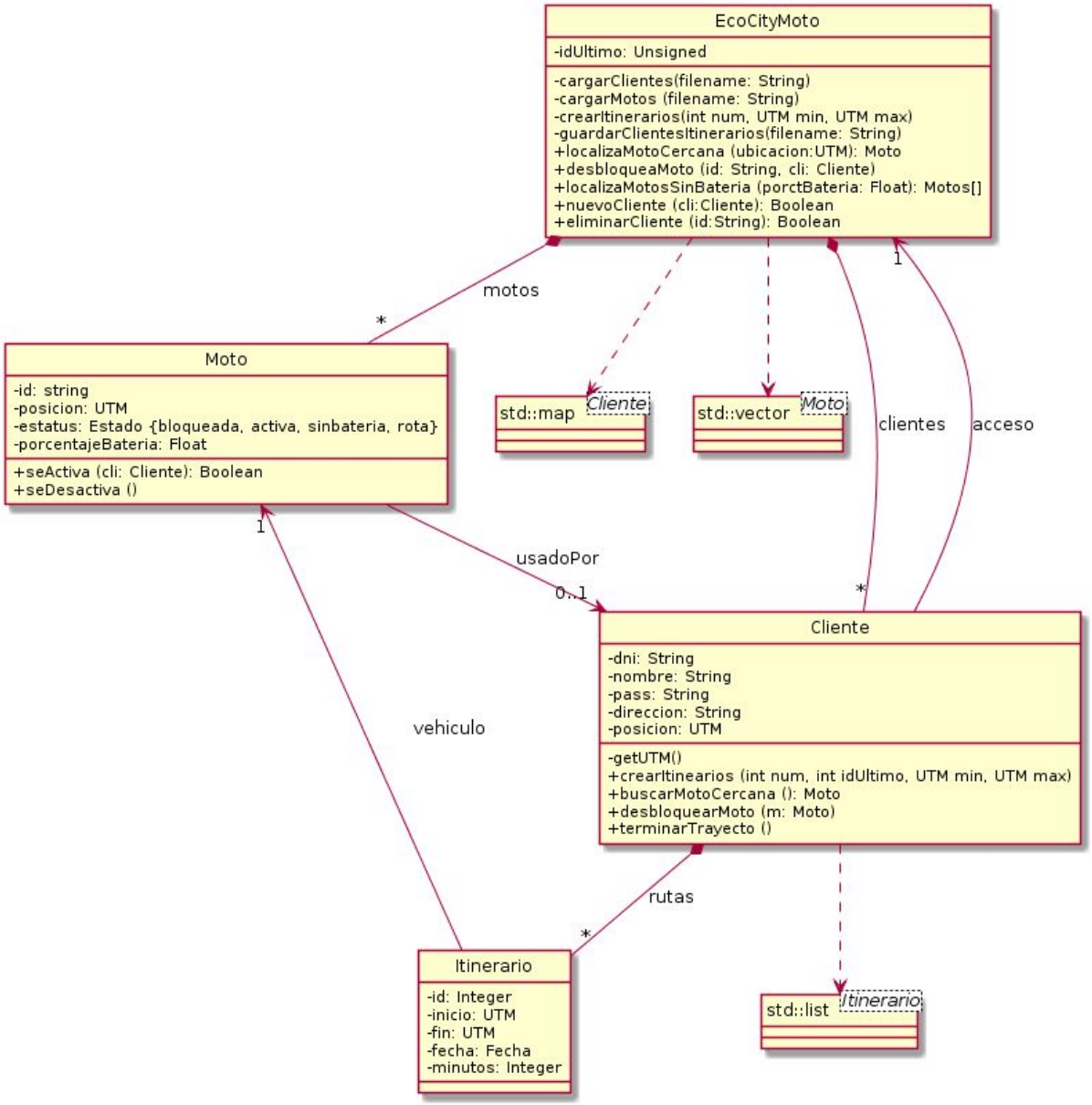
Descripción de la EEDD

En las prácticas anteriores hemos implementado unas estructuras que vienen recogidas en la biblioteca STL. En esta práctica vamos cambiar las estructuras implementadas de forma nativa según indica el nuevo esquema UML.

Diseño de la aplicación

El diseño del sistema se mantiene igual al de la práctica 3 añadiendo la siguiente funcionalidad:

- El constructor de *EcoCityMoto* debe rellenar las eedd con motos, clientes e itinerarios, y por tanto debe llamar a las funciones privadas señaladas a tal fin en dicha clase. La primera vez no se han guardado los itinerarios, así que se llama a *crearItinerararios()*, pero la siguiente vez debe estar guardado en fichero y leerse de allí. La acción de guardar debe llamarse en el destructor de la clase. Debéis decidir el formato del archivo de clientes para que, junto con los datos de cada uno, también guarde los datos de sus itinerarios en el mismo archivo que utilizó para cargarlos.
- Se añade a motos un atributo de porcentaje de batería (*porcentajeBateria*) con un valor entre [0,100]. Un minuto de funcionamiento es un 1% de descarga de batería. Si la batería está por debajo del 15% debe cambiar el estado de la moto a "*sinbatería*". El % inicial es un valor aleatorio entre 1 y 100.
- Se crea una nueva función llamada localizaMotosSinBateria() que devuelva todas las motos que estén en el estado de "*sinbatería*".



Programa de prueba

Para probar la funcionalidad de la aplicación deberá cargar los datos relativos a clientes, itinerarios y motos desde funciones dentro de *EcoCityMoto*. Tras esta operación se realizarán las siguientes operaciones:

1. Añadir a la empresa un nuevo cliente que no exista previamente con coordenadas en Jaén, rango (37, 3) - (38, 4).
2. Localizar el cliente anterior en la empresa por su DNI y buscar una moto cercana que se pueda utilizar.
3. Realizar un itinerario con la moto localizada con una duración válida para la carga de batería de la moto. Al dejarla el cliente debe comprobar la carga de la moto para poner adecuadamente el estado.

4. Localizar las motos sin batería e indicar si la moto utilizada está en esa situación.
5. Borrar el cliente que se insertó en el punto 1 y comprobar que efectivamente ya no existe.

Recordad que el destructor de la clase *EcoCityMoto* debe guardar los datos de clientes y motos.

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.