Programación Avanzada

LABORATORIO 1

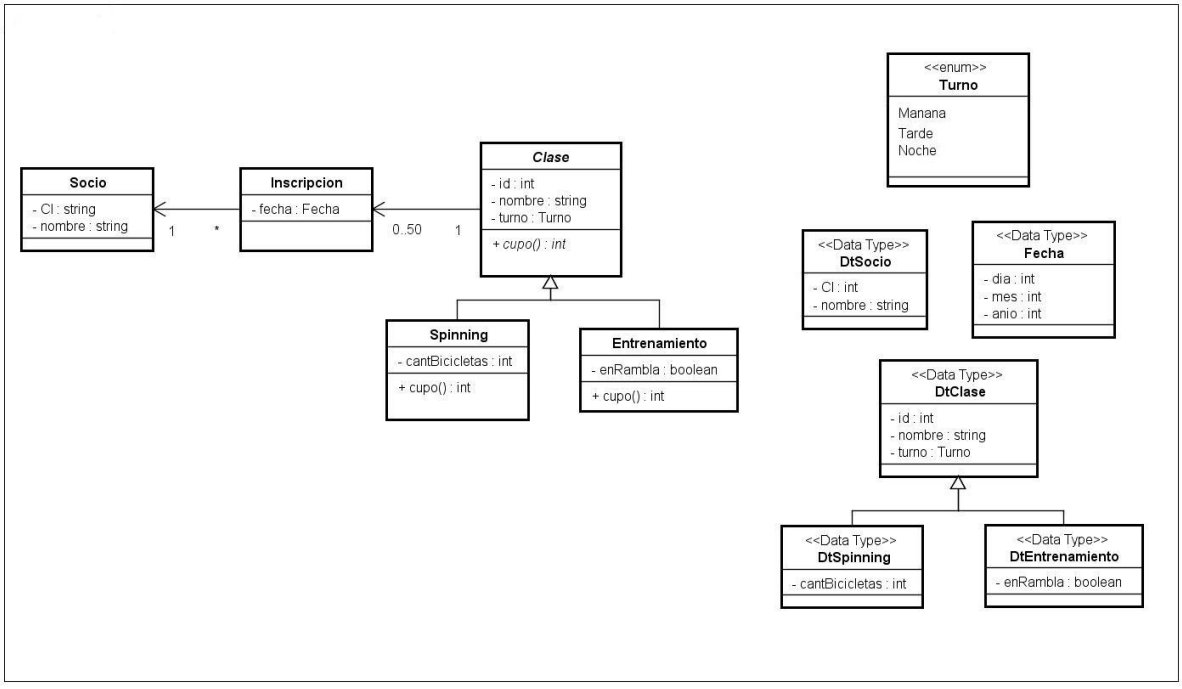
Consideraciones generales:

- La entrega podrá realizarse hasta el **Domingo 15 de abril hasta las 23:55 hs.**
- El código fuente y el archivo Makefile deberán ser entregados mediante el Campus Virtual del curso dentro de un archivo con nombre *<número de grupo>_lab1.zip*. Dentro del mismo archivo comprimido, se deberá agregar un archivo denominado *resp_lab1.txt*, con las respuestas a las preguntas planteadas, y un archivo *integrantes.txt*, con los nombres y correos electrónicos de cada uno de los miembros del grupo.
- El archivo Makefile entregado debe ser independiente de cualquier entorno de desarrollo integrado (*IDE,* por sus siglas en inglés) [1].
- Las entregas que no cumplan estos requerimientos no serán consideradas. El hecho de no realizar una entrega implica la insuficiencia del laboratorio completo.

Con este laboratorio se espera que el estudiante adquiera competencias en la implementación de operaciones básicas, el uso básico del lenguaje C++ (que se usará en el laboratorio) y el entorno de programación en Linux, así como reafirmar conceptos presentados en el curso [2,3,4,5,6]. También se espera que el estudiante consulte el material disponible en el Campus Virtual del curso y que recurra a Internet con espíritu crítico, identificando y corroborando fuentes confiables de información.

Ejercicio 1

Se desea implementar un pequeño sistema con la realidad expresada en el siguiente modelo, con el fin de manejar información sobre las inscripciones a clases de un gimnasio.



De cada inscripción se conoce el socio, la clase del gimnasio a la cual se inscribe y la fecha en que se realizó. De cada socio se conoce su cédula de identidad (que lo identifica) y su nombre. Además, de cada clase se tiene un identificador (numérico), el nombre de la clase, y el turno en que se realiza, pudiendo éste tener los valores Manana, Tarde, Noche. No puede haber dos inscripciones para una misma clase y un mismo socio.

Existen dos tipos de clases en el gimnasio: Spinning y Entrenamiento. En el caso de Spinning se conoce la cantidad de bicicletas con las que cuenta la clase, y en el caso de Entrenamiento se sabe si la clase se realiza en la rambla o no. La cantidad de bicicletas de una clase de spinning debe ser a lo sumo 50.

El sistema debe permitir calcular el cupo de cada clase, que deberá ser obtenido con la operación `cupo()`. Para las clases de Spinning, el cupo es igual a la cantidad de bicicletas. En cambio, para las clases de Entrenamiento, el cupo depende del lugar en donde se realiza la clase. Si la clase es en la rambla el cupo es de 20 personas y si no lo es, el cupo es de 10 personas. Un socio puede inscribirse a una clase únicamente si el cupo de la misma no fue alcanzado al momento de realizar la inscripción.

Se pide: Implementar en C++:

1. Todas las clases (incluyendo sus atributos, pseudoatributos, *getters*, *setters*, constructores y destructores), enumerados y *datatypes* que aparecen en el diagrama. Para las fechas en caso de recibir dd > 31 o dd < 1 o mm > 12 o mm < 1 o aaaa < 1900, se debe levantar la excepción *std::invalid_argument*. No se deben hacer más que estos controles en la fecha (ej. La fecha 31/2/2000 es válida para esta realidad).

2. Una función *main* que implemente las siguientes operaciones:
   a) `void agregarSocio(string ci, string nombre)`

      Crea un nuevo socio en el sistema. En caso de ya existir, levanta la excepción *std::invalid_argument.*

   b) `void agregarClase(DtClase& clase)`

      Crea una nueva clase en el sistema. En caso de ya existir, levanta una excepción *std*::*invalid_argument*.

   c) `void agregarInscripcion(String ciSocio, int idClase, Fecha fecha)`

      Crea una inscripción de un socio a una clase. La inscripción tiene lugar siempre y cuando el socio y la clase existan, de lo contrario se levanta una excepción *std::invalid_argument.* Si ya existe una inscripción de ese usuario para esa clase, o si el cupo de esa clase ya fue alcanzado, también se levanta una excepción *std::invalid_argument*.

   d) `void borrarInscripcion(string ciSocio, int idClase)`

      Borra la inscripción de un socio a una clase. Si no existe una inscripción de ese usuario para esa clase, se levanta una excepción *std::invalid_argument*.

   e) `DtSocio** obtenerInfoSociosPorClase (int idClase, int& cantSocios)`

      Retorna un arreglo con los socios que están inscriptos a determinada clase. El largo del arreglo de socios está dado por el parámetro cantSocios.

   f) *`DtClase& obtenerClase(int idClase)`*

      Retorna la información de la clase identificada por idClase.

Notas:

- A los efectos de este laboratorio, la función *main* mantendrá una colección de socios, implementada como un arreglo de tamaño MAX_SOCIOS. Lo mismo para las clases del gimnasio con un máximo de MAX_CLASES (ambas constantes). Puede implementar operaciones en las clases dadas en el modelo si considera que le facilitan para la resolución de las operaciones pedidas en el *main*.
- Se puede utilizar el tipo std::string para implementar los atributos de tipo string.
- El *main* debe manejar las excepciones lanzadas por las operaciones de los objetos. Por más información consulte las referencias [7]

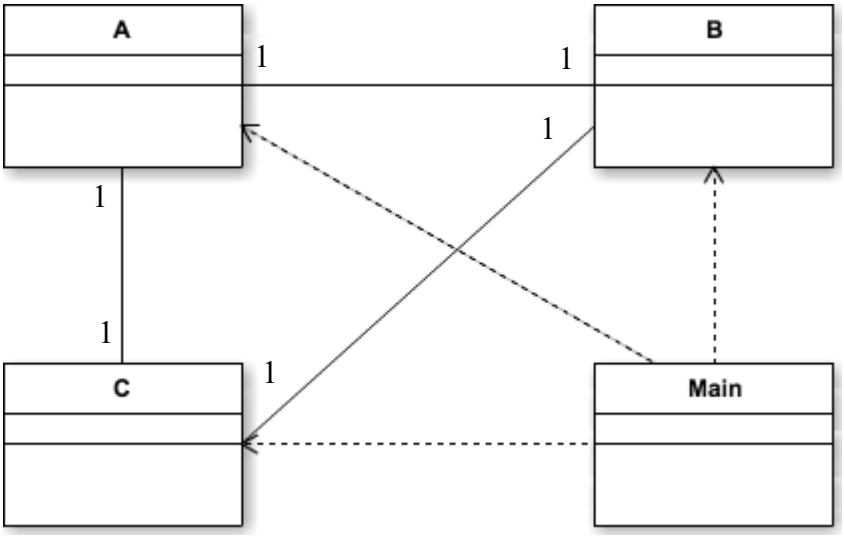
3. Sobrecargar el operador de inserción de flujo (ej. <<) en un objeto de tipo *std::ostream*. Este operador debe "imprimir" las distintas clases de DtClase (DtSpinning, DtEntrenamiento) con el siguiente formato:
Id Clase:
Nombre:

Turno (imprimiendo 'Mañana', 'Tarde' o 'Noche'):

Cantidad de bicicletas o En rambla (imprimiendo los valores 'Si' o 'No'), según corresponda

Ejercicio 2

En este ejercicio se busca que se familiarice con la problemática de dependencias circulares en C++ [8]. Para esto se debe implementar y compilar la siguiente estructura dada por el diagrama de abajo.



Se pide:

1. Implementar y compilar en C++ el diagrama dado. Defina operaciones en las clases A, B y C que impriman un mensaje con el nombre de la clase. Defina un *main* que cree objetos de esas clases e invoque a dichas operaciones.

2. Responder las siguientes preguntas:

- **a)** ¿Cuáles son las dependencias circulares que fueron necesarias solucionar para que el programa compile?
- **b)** ¿Qué es *forward declaration*?

Referencias

[1] Instructivo de Compilación.pdf

[2] Material de Programación Avanzada en el Campus Virtual, https://cv.utu.edu.uy/course/view.php?id=583

[3] Conceptos Básicos de Implementación en C++

[4] The C++ Resources Network: Information, Tutorials, Reference, Articles, Forum, http://www.cplusplus.com/doc/tutorial/

[5] Curso de C++, http://c.conclase.net/curso/

[6] Introducción a la Programación Orientada a Objetos Empleando C++ - Peter Müller,

http://www.desy.de/gna/html/cc/Tutorial/Spanish/

[7] Excepciones en C++

[8] Referencias Circulares y Namespaces