Programación 4 - 2024

Laboratorio 0 :: Conceptos Básicos en C++

Consideraciones generales:

- La entrega podrá realizarse hasta el **lunes 1° de abril de 2024 a las 15hrs.**
- El código fuente y el archivo Makefile [1] deberán ser entregados mediante el EVA del curso [2] dentro de un archivo con nombre `<número de grupo>_lab0.zip (o tar.gz)`.
- Más allá de que se sugiere el uso de un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para contar con un ambiente de desarrollo adecuado, el archivo Makefile entregado debe ser independiente de cualquier IDE, permitiendo la compilación aislada de la solución.
- El código deberá poder compilarse y ejecutarse sin errores en las máquinas Linux de la Facultad de Ingeniería.
- Las entregas que no cumplan estos requerimientos no serán consideradas. El hecho de no realizar una entrega implica la insuficiencia del laboratorio completo.

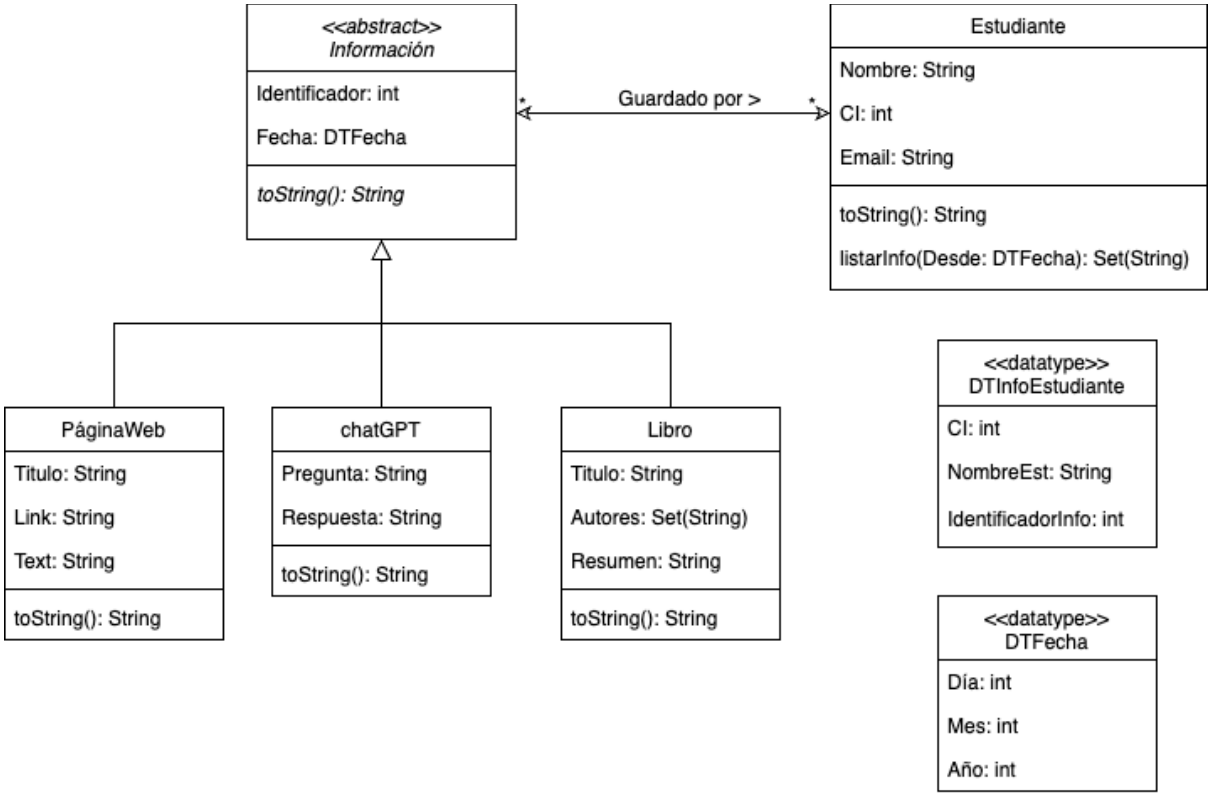
Objetivo

El objetivo del Laboratorio 0 es reforzar conceptos básicos de orientación a objetos a través de su implementación en el lenguaje C++ [3], así como practicar diversas construcciones del lenguaje y del entorno de programación en Linux [4] que serán de utilidad en etapas posteriores del laboratorio. Se espera que se consulte el material disponible en el EVA del curso (ver referencias al final de este documento), complementado con consultas a otros medios (ej.: Internet) con espíritu crítico y corroborando, en la medida de lo posible, que las fuentes consultadas sean confiables.

Problema

Se quiere construir un sistema para ayudar a los estudiantes a manejar y valorar la información que consultan mientras estudian. En el diagrama de clases UML que se muestra a continuación, existen tres clases, `PáginaWeb`, `chatGPT` y `Libro`, que son subclases de una clase abstracta `Información`. De cada información se conoce un número que la identifica y la fecha en la cual se registró esa información. Además, de las páginas web se conoce su título, la url en la cual se consultó y el texto recuperado de interés. De las consultas a chatGPT se conoce la pregunta realizada y la respuesta de ese chat de inteligencia artificial. De los libros a su vez se conoce su título, la lista de autores y un resumen. Todas estas propiedades se representan como atributos privados de las clases.

Los estudiantes son representados con la clase `Estudiante`. De cada estudiante se conoce su nombre, su cédula de identidad (que lo identifica) y su email, también representados como atributos privados de la clase. Los estudiantes pueden guardar información que consideren relevante, lo cual se representa con una asociación (`guardado por`) entre `Información` y `Estudiante`, representada con pseudoatributos en las clases correspondientes.



A nivel de comportamiento, se definen dos operaciones:

- La operación `toString()` en las clases `Información`, `PáginaWeb`, `chatGPT` y `Libro` es una operación polimórfica que devuelve un `String` con el detalle de la información del objeto al cual se le invoca la operación. Su método se define solo en las clases `PáginaWeb`, `chatGPT` y `Libro`, siendo abstracta en `Información`. El string está conformado por el tipo de objeto y el valor de todos sus atributos, separados por coma. Es decir, tienen la siguiente forma dependiendo del objeto:

  `PáginaWeb: Identificador, Fecha, Titulo, Link, Text`

  `chatGPT: Identificador, Fecha, Pregunta, Respuesta`

  `Libro: Identificador, Fecha, Titulo, Autores, Resumen`

- La operación `listarInfo(Desde: DTFecha)` de la clase `Estudiante` obtiene información de todos los registros de información con fecha anterior al parámetro `Desde`. Para ello, la operación debe iterar sobre el conjunto de informaciones asociadas al estudiante y, para cada información cuya fecha sea posterior a la incluida como parámetro en `Desde`, llamar a la operación `toString()`, acumulando todos esos string en un conjunto que es el que se retorna.

Tras implementar completamente la estructura y comportamiento descritos, es posible crear la siguiente secuencia ejecutable en un `main`.

a) Crear los siguientes objetos de la clase `PáginaWeb` (con el constructor por parámetros):

Identificador = 1
Fecha = 7/3/2024
Titulo = Programación 4 Guía Semana 1 (4/3)
Link = https://eva.fing.edu.uy/pluginfile.php/468051/mod_resource/content/4/Guia01_P42024_IntroCBasicos.pdf
Text = El objetivo de esta semana es contextualizar el paradigma de Orientación a Objetos (OO) en el marco de la Ingeniería de Software, así como comenzar a ver sus conceptos básicos y cómo éstos se implementan en C++.

Identificador = 2
Fecha = 5/3/2024
Titulo = Programación orientada a objetos
Link = https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
Text = La programación orientada a objetos (POO, en español) es un paradigma de programación que parte del concepto de "objetos" como base, los cuales contienen información en forma de campos (a veces también referidos como atributos o propiedades) y código en forma de métodos. Algunas características clave de la programación orientada a objetos son herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

b) Crear los siguientes objetos de la clase `chatGPT` (con el constructor por parámetros):

Identificador = 3
Fecha = 8/3/2024
Pregunta = ¿Qué es el polimorfismo en orientación a objetos?
Respuesta = El polimorfismo en programación orientada a objetos se refiere a la capacidad de un objeto de tomar múltiples formas. Puede ser estático, resuelto en tiempo de compilación, basado en la herencia, o dinámico, resuelto en tiempo de ejecución, asociado a interfaces o métodos abstractos. En esencia, permite que objetos de diferentes clases respondan a la misma interfaz de manera coherente, facilitando la flexibilidad y extensibilidad del código.

Identificador = 4
Fecha = 5/3/2024
Pregunta = ¿Qué es el acoplamiento en orientación a objetos?
Respuesta: El acoplamiento en programación orientada a objetos se refiere al grado de dependencia entre las clases o módulos de un sistema. Un bajo acoplamiento es deseable, ya que implica que las clases son independientes entre sí, lo que facilita la modificación, mantenimiento y reutilización del código. Por otro lado, un alto acoplamiento indica una fuerte interdependencia entre las clases, lo que puede hacer que el sistema sea más difícil de entender, modificar y mantener.

c) Crear los siguientes objetos de la clase `Libro` (con el constructor por parámetros):

Identificador = 5
Fecha = 15/3/2024
Titulo = Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development
Autores = Craig Larman
Resumen = Applying UML and Patterns is the world's #1 business and college introduction to "thinking in objects"—and using that insight in real-world object-oriented analysis and design. Building on two widely acclaimed previous editions, Craig Larman has updated this book to fully reflect the new UML 2 standard, to help you master the art of object design, and to promote high-impact, iterative, and skillful agile modeling practices.

d) Consultar los objetos creados invocando la operación `toString()` en cada uno de ellos.

e) Crear los siguientes objetos de la clase `Estudiante` (con el constructor por parámetros):

Nombre = Alex García
CI = 52365899
Email = ag5678@gmail.com

Nombre = Betina Gonzalez
CI = 49891239
Email = beg999@gmail.com

f) Registrar las siguientes relaciones entre estudiantes e informaciones (creando links de la relación en ambas direcciones)

- Estudiante = Alex García Información = 1
- Estudiante = Alex García Información = 2
- Estudiante = Alex García Información = 3
- Estudiante = Betina Gonzalez Información = 3
- Estudiante = Betina Gonzalez Información = 4
- Estudiante = Betina Gonzalez Información = 5

g) Consultar la información registrada invocando la operación `listarInfo("8/3/2024")` para cada uno de los estudiantes.

h) Buscar toda la información que contenga el término "polimorfismo" y listar los estudiantes que la han guardado para leer, creando un set de `DTInfoEstudiante` con lo encontrado. Para esto se debe recorrer el conjunto de objetos de la clase `Información` y utilizar la operación `toString` para obtener un string que represente el objeto, para luego buscar en él el término. Para cada objeto que cumpla que el término "polimorfismo" se encuentra en su información, se recorren los estudiantes relacionados con ese objeto `Información`. Para

cada par <Información, Estudiante> encontrado se crea una DTInfoEstudiante y se guarda en una lista, la cual finalmente es el retorno.

i) Eliminar un objeto de la clase Información. Para ello es necesario, no solo invocar al destructor del objeto, sino también eliminar todos los links de la relación que vinculen a ese objeto con un Estudiante. Por ejemplo, si se elimina el objeto Información = 3, al buscar la información que contenga el término "polimorfismo", el objeto no debe aparecer.

Se pide:

1. Implementar todas las clases (incluyendo sus atributos, pseudo-atributos, constructores, destructores, getters y setters) y datatypes que aparecen en el diagrama.
2. Sobrecargar el operador de inserción de flujo (<<) en un objeto de tipo std::ostream. Este operador debe permitir "imprimir" todos los datos del datatype DTInfoEstudiante, siguiendo un formato similar al siguiente:

   ```
   52365899, Alex García, 5
   ```

3. Implementar la operación toString() en las clases Información, PáginaWeb, chatGPT y Libro.
4. Implementar la operación listarInfo(Desde: DTFecha): Set(String) en la clase Estudiante.
5. Implementar un método main que defina un conjunto de Información y un conjunto de Estudiante y que realice la secuencia ejecutable definida previamente, en donde las actividades de consulta de la secuencia (d, g y h) deben tener salida a consola, incluidas las posteriores a la eliminación de un objeto.

Notas:

- Puede implementar operaciones auxiliares en las clases dadas en el diagrama si considera que facilitan la resolución de las operaciones pedidas.
- Se puede utilizar el tipo std::string [5] para implementar los atributos de tipo string, así como estructuras de datos de la biblioteca STL [6], tales como vector, set, map, etc.
- Se debe solucionar problemas de dependencias circulares entre las clases, por ejemplo con relaciones bidireccionales. Para ello es necesario utilizar declaraciones en avanzada (forward declarations) [7].

Referencias

[1] Programación 4. Instructivo de Compilación
URL: https://eva.fing.edu.uy/course/view.php?id=413&section=3

[2] EVA Programación 4. URL: https://eva.fing.edu.uy/course/view.php?id=413

[3] C++. URL: https://www.cplusplus.com/

[4] Unidad de Recursos Informáticos, Salas Linux.
URL: https://www.fing.edu.uy/es/sysadmin/ensenanza/salas-linux

[5] Tipo `std:: string`
URL: https://en.cppreference.com/w/cpp/string/basic_string

[6] C++ Standard Template Library (STL)
URL: https://cplusplus.com/reference/stl/

[7] Programación 4. Referencias Circulares y Namespaces
URL: https://eva.fing.edu.uy/course/view.php?id=413&section=3