Programación 4 - 2023

Laboratorio 0 :: Conceptos Básicos en C++

Consideraciones generales:

- La entrega podrá realizarse hasta el **lunes 20 de marzo de 2023 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 ejecutarse sin errores de compilación en las máquina 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.

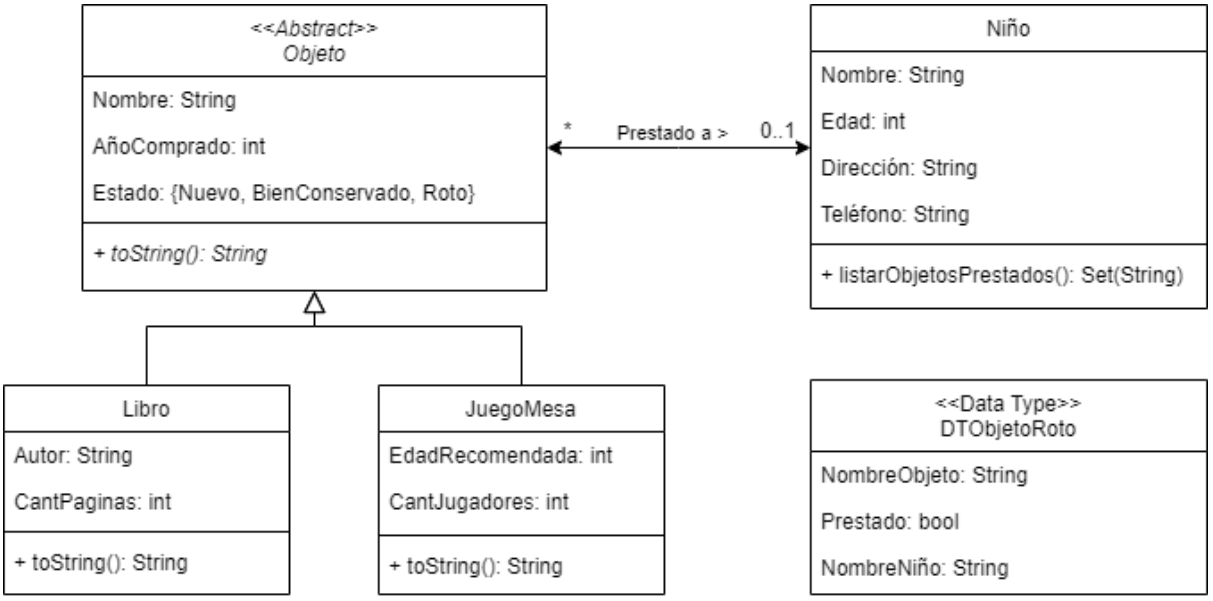
Objetivo

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 una biblioteca comunitaria en la cual los niños pueden pedir prestado libros y juegos de mesa. En el diagrama de clases UML que se muestra a continuación, existen dos clases `Libro` y `JuegoMesa` que son subclases de una clase abstracta `Objeto`. De cada objeto se conoce su nombre (que lo identifica), el año en el que se compró, su estado (un enumerado con valores nuevo, bien conservado y roto). Además, de los libros se conoce su autor y cantidad de páginas, mientras que de los juegos de mesa se conoce la edad a partir de la cual se recomienda su uso y la cantidad de jugadores mínimos. Todas estas propiedades se representan como atributos privados de la clase. Los niños son representados con una clase `Niño`, de los que se conoce su nombre (que lo identifica), edad, dirección y número de teléfono de contacto, también representados como atributos privados de la clase. Los niños pueden pedir prestados objetos, lo cual se representa con una asociación (`prestado a`) entre objeto y niño, representada con pseudoatributos en las clases correspondientes. En el diagrama se expresa que un objeto puede o no tener un niño vinculado (multiplicidad `0..1`) dependiendo de si está o no prestado, así como que un niño puede o no tener varios objetos en préstamo (multiplicidad `*`). Finalmente, se define un datatype `DTObjetoRoto`

que permite informar qué objetos están rotos a través del nombre del objeto, la identificación si está prestado o no y el nombre del niño en caso de estar prestado.



A nivel de comportamiento, se definen dos operaciones:

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

  `Libro: Nombre, AñoComprado, Estado, Autor, CantPaginas`

  `Juego: Nombre, AñoComprado, Estado, EdadRecomendada, CantJugadores`

- La operación `listarObjetosPrestados()` de la clase `Niño` obtiene información de todos los objetos prestados a un niño en particular. Para ello, la operación debe iterar sobre el conjunto de objetos asociados al niño y, para cada objeto, 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 `Libro` (con el constructor con parámetros):

Nombre = Nacidos de la bruma: El imperio final
AñoComprado = 2022
Estado = Roto
Autor = Brandon Sanderson
Cantpaginas = 688

Nombre = Las Malas
AñoComprado = 2022
Estado = Nuevo
Autor = Camila Sosa Villada
Cantpaginas = 240

Nombre = El cocodrilo al que no le gustaba el agua
AñoComprado = 2016
Estado = Roto
Autor = Gemma Merino
Cantpaginas = 32

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

Nombre = Juego Uno
AñoComprado = 2022
Estado = Roto
EdadRecomendada = 7
CanJugadores = 10

Nombre = Mazo de Cartas
AñoComprado = 2019
Estado = Nuevo
EdadRecomendada = 7
CanJugadores = 4

Nombre = Dados
AñoComprado = 2020
Estado = Roto
EdadRecomendada = 2
CanJugadores = 6

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

d) Crear los siguientes objetos de la clase `Niño` (con el constructor con parámetros):

Nombre = María Laura
Edad = 10
Dirección = Nueva Palmira 1521
Teléfono = 099298190

Nombre = Alex
Edad = 5
Dirección = Humberto Primo 1501
Teléfono = 29094141

e) Registrar los siguientes préstamos (creando links de la relación en ambas direcciones)
   - Niño = María Laura Objeto = Mazo de Cartas
   - Niño = María Laura Objeto = Nacidos de la bruma: El imperio final
   - Niño = María Laura Objeto = Dados
   - Niño = Alex Objeto = Juego Uno
   - Niño = Alex Objeto = El cocodrilo al que no le gustaba el agua

f) Consultar los préstamos invocando la operación `listarObjetosPrestados()` para cada uno de los niños.

g) Consultar los objetos rotos recorriendo el conjunto de objetos de la clase `Objeto` y generando elementos del datatype `DTObjetoRoto` para cada objeto roto. Es decir, si se encuentra un objeto cuyo valor del atributo `Estado` es igual a `Roto`, se construye un datavalue del datatype `DTObjetoRoto` con el nombre del objeto. Además, si el objeto está asociado a un niño (o sea, que está prestado), el campo `Prestado` del datavalue se pone en `true` (`false` en caso contrario) y se recupera el nombre del objeto niño vinculado que se almacena en el campo `NombreNiño` del datavalue.

h) Eliminar un objeto de la clase `Objeto`. 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 niño. Por ejemplo, si se elimina el objeto Objeto = Juego Uno, al consultar los objetos prestados de Alex, el objeto no debe aparecer. Tampoco debe aparecer si se consulta nuevamente la lista de objetos rotos.

Se pide:

1. Implementar todas las clases (incluyendo sus atributos, pseudo-atributos, constructores, destructores, getters y setters), enumerados 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 `DTObjetosRotos`, siguiendo un formato similar al siguiente:

   `NombreObjeto, Prestado SI/NO, NombreNiño` (si Prestado SI)

   Por ejemplo:
   ```
   Mazo de Cartas, Prestado SI, María Laura
   Las Malas, Prestado NO
   ```

3. Implementar la operación `toString()` en las clases `Objeto`, `Libro` y `JuegoMesa`.

4. Implementar la operación `listarObjetosPrestados()` en la clase `Niño`.

5. Implementar un método `main` que defina un conjunto de `Objeto` y un conjunto de `Niño` y que realice la secuencia ejecutable definida previamente, en donde las actividades de consulta de la secuencia (c, f, 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 le facilitan para 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 bidirecciones. 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