E2. Proyecto Integrador

Entrega de la Situación Problema

Miembros del equipo:

Karol Alexis Alvarado Davila – A01751711 - IDM

César Pascual De La Torre - A01751521 - IDM

Profesor: Carlos Daniel Prado Pérez

Programación orientada a objetos

TC1030.302

Instituto Tecnológico de Monterrey, Campus Estado de México

Lunes 10 de junio del 2024

Apegándome a la Integridad Académica de los Estudiantes del Tecnológico de Monterrey, me comprometo a que mi actuación en esta actividad esté regida por la integridad académica. En congruencia con el compromiso adquirido, realizaré este trabajo de forma honesta y personal, para reflejar, a través de él, mi conocimiento y aceptar, posteriormente, la evaluación obtenida.

Firmas:

Índice de contenido

Introducción.....3

UML.....4

Ejemplos de ejecución.....6

Argumentación.....8

Identificación de casos que harían que el proyecto deje de funcionar.....8

Conclusiones Karol Alexis Alvarado Davila.....8

Conclusiones César Pascual De La Torre 9

Referencias 11

Introducción

El problema a resolver es el desarrollo de un sistema para un servicio de streaming de video bajo demanda que maneje dos tipos de videos: películas y series. Cada video debe tener un ID, un nombre, una duración y un género (drama, acción, misterio). Además, las series deben incluir episodios, cada uno con un título y una temporada correspondiente. Es fundamental conocer la calificación promedio de cada video, en una escala de 1 a 5, y el sistema debe ser capaz de mostrar videos y episodios con sus calificaciones.

Tendremos que diseñar e implementar un sistema que utilice conceptos de programación orientada a objetos, como herencia para compartir atributos comunes entre series y películas, polimorfismo para manejar objetos de diferentes tipos uniformemente, y sobrecarga de operadores para definir comportamientos específicos al interactuar con los objetos. Además, la aplicación deberá gestionar información sobre diferentes tipos de videos y generar reportes como películas de un cierto género, series de un cierto género y películas con sus calificaciones.

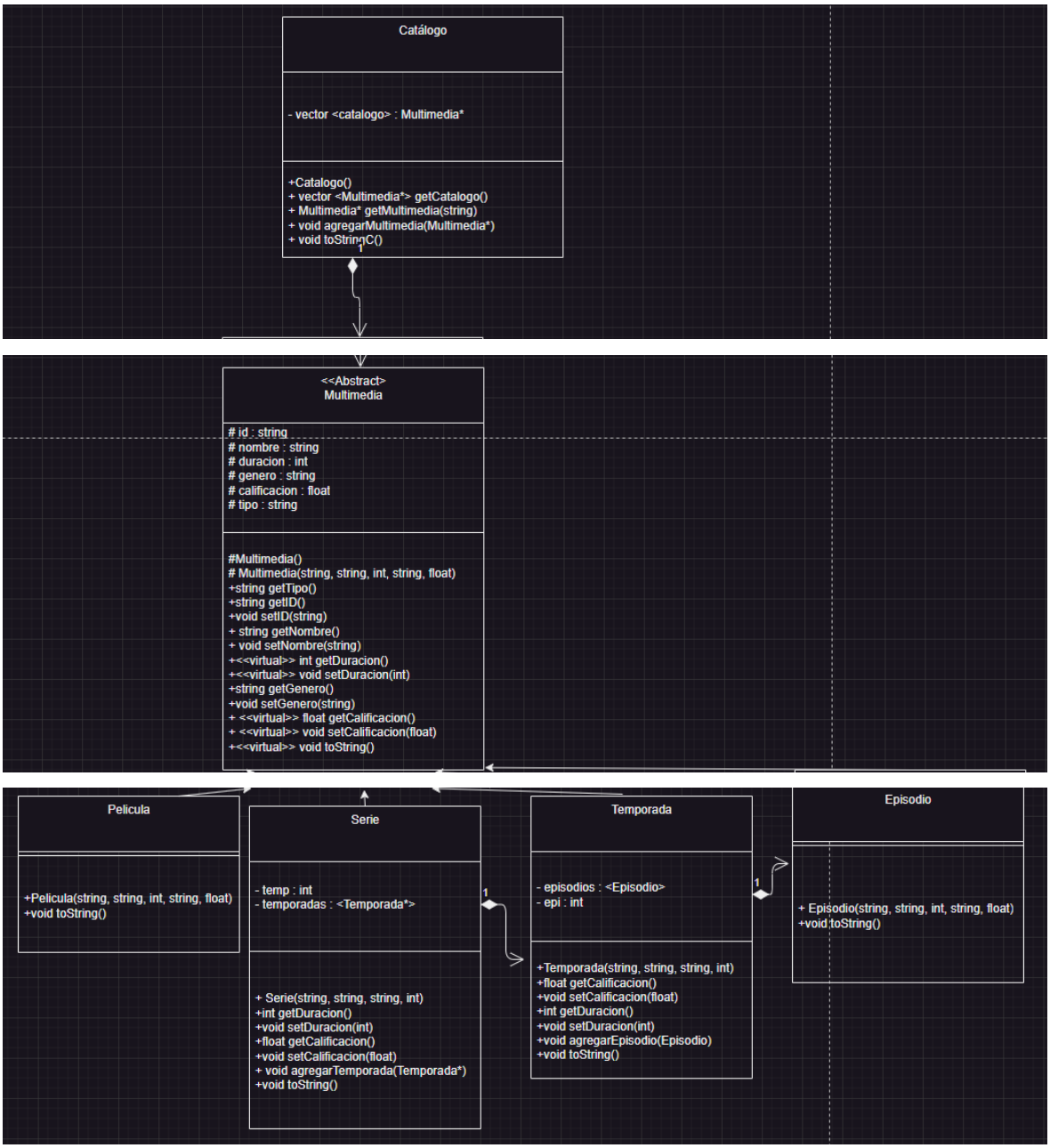
El desarrollo incluirá un modelo de clases con herencia y polimorfismo, representado en un Diagrama UML, y su implementación en un lenguaje de programación. La aplicación tendrá una interfaz de usuario con un menú ciclado para cargar datos, mostrar videos y episodios con ciertas calificaciones o géneros, calificar videos y salir. Será crucial validar todas las entradas del usuario para asegurar la precisión y funcionalidad del sistema, proporcionando así una solución eficiente para la gestión y evaluación de contenidos en un servicio de streaming.

UML

Para el diagrama UML de las clases involucradas para nuestra propuesta de solución consideramos lo siguiente:

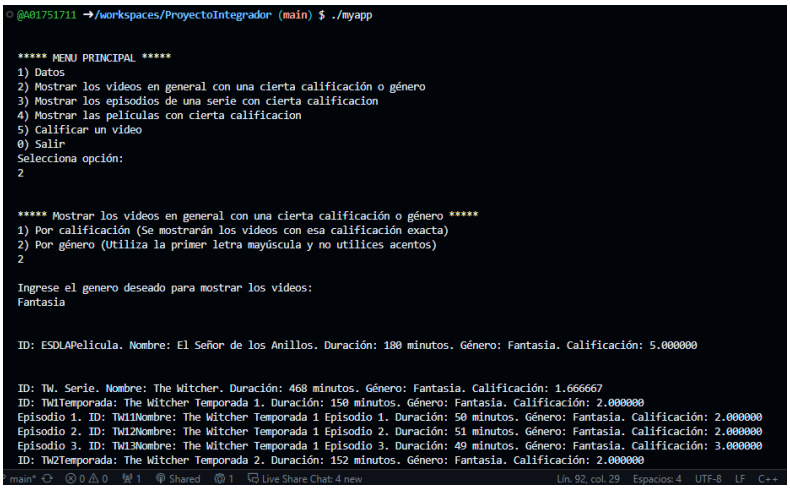
- Catálogo: Objeto que almacena los objetos de tipo multimedia.
- Multimedia: Clase abstracta base para contenido multimedia (id, duración, calificación, etc.). Con sus respectivos métodos virtuales que son reescritos en subclases.
    - Película: Objeto de tipo multimedia que reescribe el método “toString()”.
    - Serie: Objeto de tipo multimedia que reescribe métodos relacionados con duración (suma de temporadas), calificación (promedio de temporadas), etc.
    - Temporada: Objeto de tipo multimedia que reescribe métodos relacionados con duración (suma de episodios), calificación (promedio de episodios), etc.
    - Episodio: Objeto de tipo multimedia que reescribe el método toString().

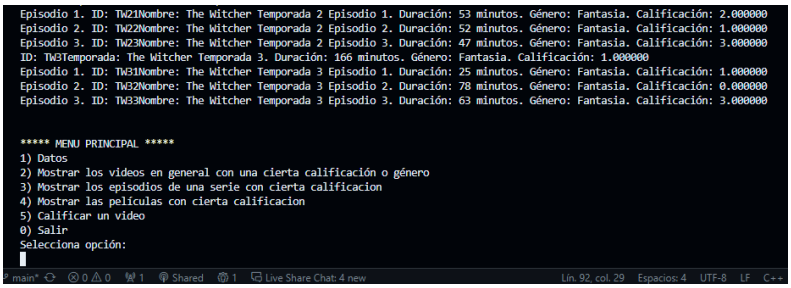
La clase Catálogo *tiene* a Multimedia, que a su vez *hereda* a 4 clases hijas: Película, Serie, que *tiene* a Temporada, que a su vez *tiene* a Episodio.


Catálogo
- vector <catalogo> : Multimedia*
+Catalogo()
+ vector <Multimedia*> getCatalogo()
+ Multimedia* getMultimedia(string)
+ void agregarMultimedia(Multimedia*)
+ void toStringC()
1
<<Abstract>>
Multimedia
id : string
nombre : string
duracion : int
genero : string
calificacion : float
tipo : string
#Multimedia()
Multimedia(string, string, int, string, float)
+string getTipo()
+string getID()
+void setID(string)
+ string getNombre()
+ void setNombre(string)
+<<virtual>> int getDuracion()
+<<virtual>> void setDuracion(int)
+string getGenero()
+void setGenero(string)
+ <<virtual>> float getCalificacion()
+ <<virtual>> void setCalificacion(float)
+<<virtual>> void toString()
Pelicula
+Pelicula(string, string, int, string, float)
+void toString()
Serie
- temp : int
- temporadas : <Temporada*>
+ Serie(string, string, string, int)
+int getDuracion()
+void setDuracion(int)
+float getCalificacion()
+void setCalificacion(float)
+ void agregarTemporada(Temporada*)
+void toString()
1
Temporada
- episodios : <Episodio>
- epi : int
+Temporada(string, string, string, int)
+float getCalificacion()
+void setCalificacion(float)
+int getDuracion()
+void setDuracion(int)
+void agregarEpisodio(Episodio)
+void toString()
1
Episodio
+ Episodio(string, string, int, string, float)
+void toString()

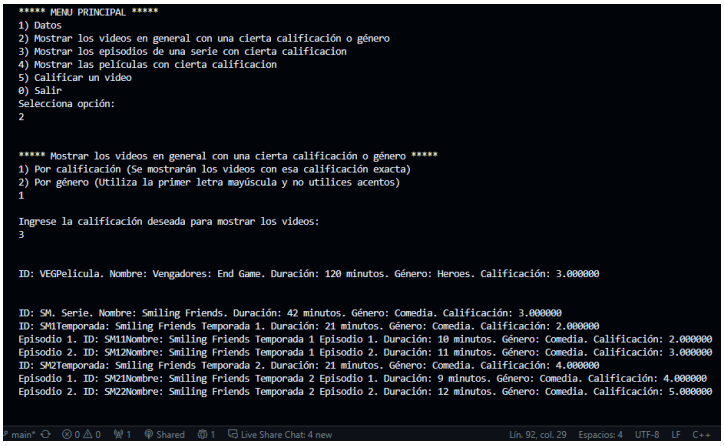
Ejemplos de ejecución

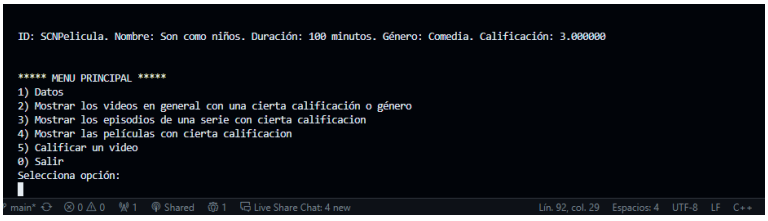
Mostrar videos por género: Fantasía



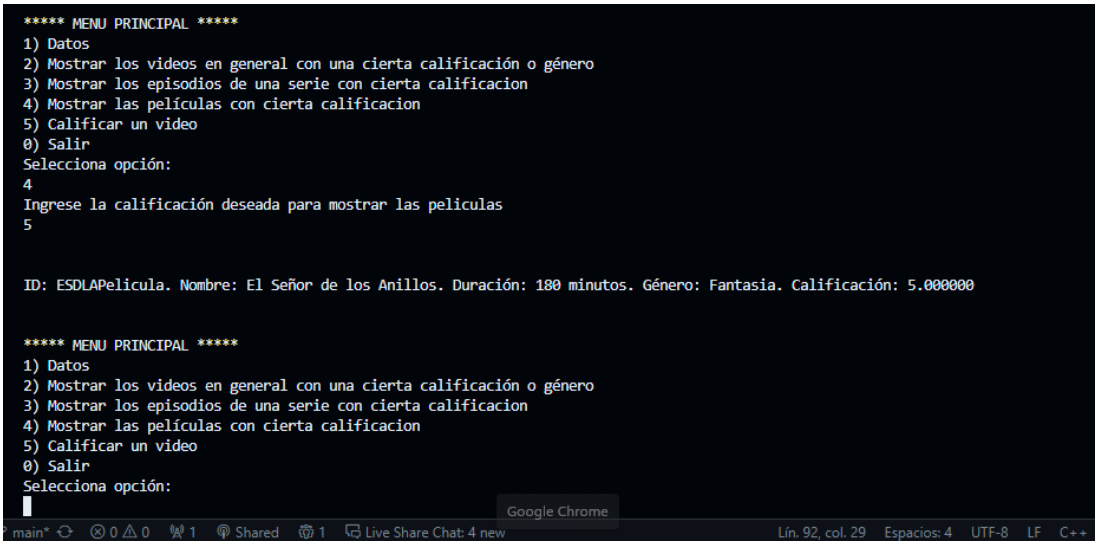


Mostrar videos por Calificación: 3

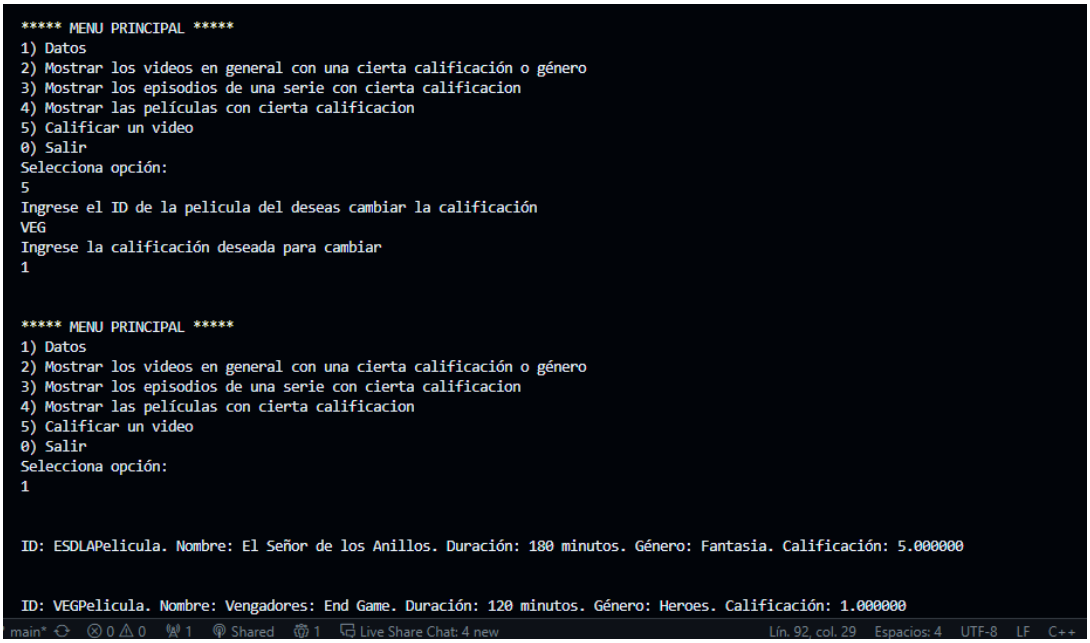




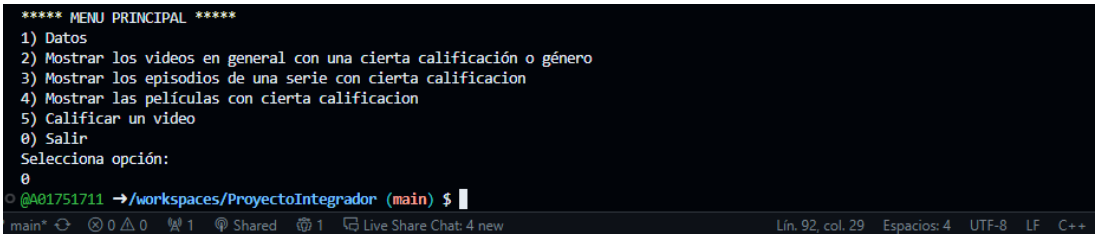
Mostrar películas con cierta calificación: 5



Calificar un video: 1



Salir



Argumentación

Para cada opción del menú, se creó un método void que realizaba las acciones necesarias tomando como parámetro siempre un apuntador que en el método main se asignó al catálogo con todos los objetos de tipo multimedia.

La mayoría de los métodos tienen ciclos que recorren un vector local a cada método que es una copia del catálogo y según el método se llaman a imprimir o cambiar alguna variable. El uso correcto de apuntadores fue clave para poder estar accionando con el objeto multimedia del catalogo en main.

Los métodos virtuales escogidos fueron los relacionados con duración y calificaciones ya que estos son los que cambian entre ellos, ya que el género es el mismo no importa el tipo de multimedia, pero las calificación es un promedio dependiendo del tipo y la duración es la suma del vector local del objeto que se esté instanciando.

Se utilizó la comprobación de errores en los inputs de los menús existentes en el codigo para asegurarse de que el usuario utilizo caracteres enteros para sus selecciones.

Identificación de casos que harían que el proyecto deje de funcionar

Una situación que haría que el código dejará de funcionar sería un error de clasificación de objetos de tipo “Serie” y “Película”. Este error como tal no haría que el proyecto dejará de funcionar, pero si provocaría errores en la creación de los episodios de la Serie clasificada como Película.

Conclusión Karol Alvarado

Personalmente, este proyecto me ha proporcionado una perspectiva mucho más clara y profunda sobre las relaciones conceptuales en la programación. En cursos anteriores, mi enfoque principal era simplemente escribir líneas de código para resolver problemas específicos. Sin embargo, a lo largo de este proyecto, he aprendido a valorar la importancia de las relaciones de herencia y composición en el diseño de software, así como la utilidad de los apuntadores y otros conceptos avanzados en C++.

Uno de los aspectos más reveladores fue entender correctamente las relaciones entre clases. Por ejemplo, en lenguaje natural, podría parecer lógico que una serie sea una clase hija de

temporada, que temporada sea una clase hija de serie, y que serie sea una clase hija de multimedia. No obstante, este proyecto me ayudó a identificar que este no es el caso. En realidad, una temporada no es un tipo de serie; más bien, una serie contiene temporadas. Esta comprensión precisa de las relaciones entre clases es fundamental para diseñar estructuras de datos coherentes y eficientes.

Además, el proyecto me ha permitido aplicar nuevos métodos para optimizar el código. Un ejemplo claro de esto es el uso de bloques try y catch para el manejo de excepciones en lugar de depender únicamente de cadenas de ifs para verificar datos. Este enfoque no solo hace el código más limpio y manejable, sino que también mejora su eficiencia y robustez.

Otra área en la que he profundizado es en la utilización de apuntadores. Antes, mi uso de apuntadores era bastante básico, pero ahora entiendo mejor su importancia y cómo pueden utilizarse para gestionar la memoria de manera efectiva, mejorar el rendimiento del programa y facilitar la implementación de estructuras de datos complejas como listas enlazadas y árboles.

En resumen, este proyecto final de C++ no solo ha ampliado mis habilidades técnicas, sino que también ha transformado mi manera de pensar y abordar los problemas de programación. Me ha enseñado a ver más allá de la mera escritura de código, enfocándome en el diseño conceptual y en las mejores prácticas de programación, lo cual es crucial para el desarrollo de software de calidad.

Conclusión César Pascual

Tras finalizar este proyecto, puedo mirar atrás e identificar varios aprendizajes en cuanto a programación orientada a objetos, como también en el trabajo en equipo. Realizar ese trabajo fortaleció mi conocimiento en lenguaje de programación y amplió mi manera de atender un trabajo en equipo, priorizando la colaboración

Uno de los principales retos que enfrentamos al realizar este proyecto fue identificar la jerarquía de clases y herencia que queríamos utilizar. Personalmente, aprendí que a pesar de que una relación de herencia pueda facilitar el código y la compresión de este, no siempre es correcto conceptualmente implementarlo de esta manera. Es importante mantener este principio en cara a mi carrera profesional ya que puede impactar mucho en el entendimiento y colaboración de un código.

De la misma manera, el aprendizaje que me llevó en cuanto a entender y plantear una situación con lenguaje orientado a objetos es muy valioso. Poder identificar una situación problema y plantearla para facilitar su solución es muy importante para un programador. De la misma manera, mi interpretación y abstracción de un código mejoró exponencialmente.

Asimismo, me llevo una experiencia colaborativa que me abrió los ojos acerca de cómo se ve un trabajo en equipo eficiente. Normalmente en proyectos anteriores solía dividir el proyecto en partes individuales para distribuir la carga y optimizar el tiempo. Sin embargo, he dado cuenta de que realizar un proyecto colaborativo y unísona resulta mucho más efectivo y reduce drásticamente el tiempo de corrección de errores de código, y aumenta el conocimiento obtenido.

Finalmente, puedo decir que realizar este proyecto desarrolló muchas de las competencias necesarias para ser un practicante profesional de mi carrera. Habilidades claves como el trabajo colaborativo, la abstracción y el entendimiento de un lenguaje de programación orientado a objetos me hacen sentir más preparado cada día como científico de datos.

Referencias

Agarwal, H. (2023). *C++ Polymorphism*. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/cpp-polymorphism/

Agarwal, H. (2024). *Inheritance in C++*. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/inheritance-in-c/

GeeksForGeeks. (2024). *Exception Handling in C++.*

Recuperado de: https://www.geeksforgeeks.org/exception-handling-c/

GeeksForGeeks. (2024). *C Pointers.*

Recuperado de: https://www.geeksforgeeks.org/c-pointers/

GeeksForGeeks. (2023). *Operator Overloading in C++.*

Recuperado de: https://www.geeksforgeeks.org/operator-overloading-cpp/

GeeksForGeeks. (2024). *References in C++.*

Recuperado de: https://www.geeksforgeeks.org/references-in-cpp/

Kariya, A. (2024). *C++ Pointers*. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/cpp-pointers/

Kartik. (2024). Applications of Pointers in C. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/applications-of-pointers-in-c-cpp/

Karunakar, V. (2024). *Object Oriented Programming in C++.* GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/

Raj, R. (2023). *Pointers vs References in C++*. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/pointers-vs-references-cpp/

Rani, B. (2023). *OOPs | Object Oriented Design*. GeeksForGeeks.

Recuperado de: https://www.geeksforgeeks.org/oops-object-oriented-design/