Tecnológico de Monterrey

Campus Monterrey

Integrative Project | Streaming service modeling

Gilberto Malagamba Montejo - A00839075
Maria Luisa Ramos Contreras - A00838813
Sara Sofía Hernández Valdez - A01801003

Juan Francisco Perez Rangel

Object-Oriented Programming
TC1030
307

June 14th , 2024

Index

1. Introduction.

Streaming platforms have become a cornerstone of modern entertainment, significantly changing traditional viewing habits. According to recent statistics, streaming services have surpassed traditional cable TV in terms of viewership and subscribers. As of 2023, Netflix alone boasts more than 240 million subscribers worldwide, while Disney+ has rapidly grown to more than 160 million subscribers since its launch in 2022. These numbers underscore the massive shift toward digital streaming as the preferred method of content consumption.

The rise of streaming platforms can be attributed to several key factors:

- Convenience: Users can watch content anytime, anywhere, on multiple devices, without the constraints of a fixed broadcast schedule.
- Personalization: Advanced algorithms provide personalized recommendations, increasing user engagement and satisfaction.
- Variety and exclusivity: Platforms offer a wide variety of content, including exclusive originals that attract and retain subscribers.
- Cost-effectiveness: Subscription-based models are often more economical than traditional cable packages, offering better value for money.

Based on the above, we can say that in recent years, the entertainment industry has undergone a significant transformation due to the emergence of low-cost, on-demand streaming services, such as Netflix, Disney+, and Prime Video. These platforms have fundamentally altered the manner in which audiences consume media, offering vast libraries of content that can be accessed at any time. Some services prioritize the provision of a large volume of videos in order to attract users, while others focus exclusively on their own branded content. This trend highlights the crucial role of efficient content management systems capable of handling diverse video libraries.

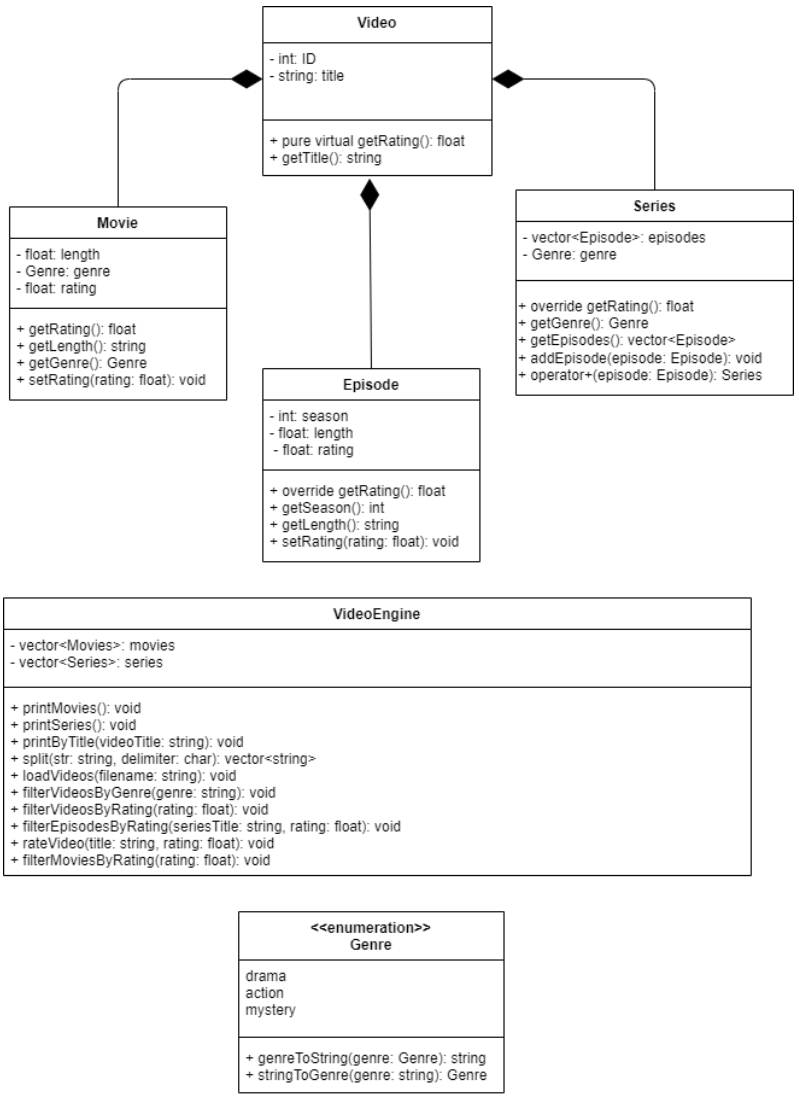
The development of user interfaces (UI) for streaming platforms is crucial in delivering a seamless and enjoyable user experience. A well-designed interface can significantly impact user retention and engagement. Key aspects include:

- User-Friendly Navigation: Easy-to-navigate interfaces help users find and enjoy content quickly, reducing frustration and improving satisfaction.
- Search and Discovery: Efficient search functions and discovery tools, such as genre-based browsing and curated recommendations, enhance content accessibility.
- Content Management: Robust systems for managing and displaying content, including ratings and reviews, provide users with valuable information to guide their viewing choices.

- Performance and Reliability: A smooth, responsive UI ensures that users can stream content without interruptions or delays, maintaining a positive experience.

This document outlines the development of a prototype for a future content provider tailored to these streaming services. The prototype will manage two types of videos: movies and series. Each video will have a unique ID, name, length, and genre (e.g., drama, action, mystery). For series, additional details such as episodes, titles, and seasons will be managed. A significant aspect of this system is its capacity to monitor and present the average user ratings for each video on a scale of 1 to 5, with 5 representing the highest rating.

2. UML class diagram with a textual justification of the design.



The provided UML Class Diagram outlines the structure of a system designed to manage a collection of videos, including movies and series, for a streaming service.

The Video class is an abstract base class representing a generic video. It includes common attributes such as int ID and string title, and declares a pure virtual function float getRating() to enforce the implementation in derived classes. The string getTitle() method is also provided to retrieve the video title.

The Movie class inherits from the Video class, extending it with attributes specific to movies such as float length, Genre genre, and float rating. It overrides the getRating() method and introduces additional methods like string getLength(), Genre getGenre(), and void setRating(float) to manage movie-specific data.

The Series class, also derived from Video, represents a TV series and contains a collection of Episode objects. It has attributes like vector<Episode> episodes and Genre genre, and overrides the getRating() method. Additional methods such as Genre getGenre(), vector<Episode> getEpisodes(), void addEpisode(Episode), and Episode getEpisode(int) facilitate the management of series episodes. The operator= is overloaded to handle episode assignments.

The Episode class represents an individual episode of a series. It includes attributes int season, float length, and float rating, and overrides the getRating() method. Methods such as int getSeason(), float getLength(), and void setRating(float) are provided for episode-specific operations.

The VideoEngine class manages collections of Movie and Series objects. It includes attributes vector<Movie> movies and vector<Series> series, and methods for various operations like void printMovies(), void printSeries(), void printByTitle(string), and filtering methods (void filterVideosByGenre(string), void filterVideosByRating(float), etc.). This class centralizes the functionality needed to manage and display video collections effectively.
The Genre enumeration defines possible genres (drama, action, mystery) and includes methods string GenreToString(Genre) and Genre StringToGenre(string) to facilitate conversions between string representations and enumeration values

The structure of the design supports scalability and maintainability, which are critical for a growing system like a streaming service. The use of a base class (Video) with derived classes (Movie and Series) allows for easy extensibility. New video types can be added by creating new subclasses of Video without changing existing code, ensuring scalability.

Encapsulation and polymorphism contribute to maintainability by reducing code duplication and improving readability. Any changes made to the base class are automatically propagated to subclasses, minimizing redundant code changes and ensuring consistency.

Centralizing video management in the VideoEngine class adds to the flexibility of the design. By providing a clear interface for video-related operations, the system can easily extend its functionality as needed. This approach ensures that the video management logic is consolidated in one place, simplifying maintenance and future development.

3. Execution example.

```
PS C:\Users\tacol\Documents\github repos\Proyecto-Final-TC1030\src> .\executable.exe

Menu:
1. Show all movies
2. Show all series
3. Show videos with a specific rating or genre
4. Show episodes of a specific series with a specific rating
5. Show movies with a specific rating
6. Rate a video
7. Exit
Enter your choice: 1
movieExample1 (Drama) - 4.9/5 - 1:30
movieExample2 (Mystery) - 3.5/5 - 2:20
Type y to continue: y

Menu:
1. Show all movies
2. Show all series
3. Show videos with a specific rating or genre
4. Show episodes of a specific series with a specific rating
5. Show movies with a specific rating
6. Rate a video
7. Exit
Enter your choice: 6
Enter title of the video: movieExample1
Enter rating: 3.5
Type y to continue: y

Menu:
1. Show all movies
2. Show all series
3. Show videos with a specific rating or genre
4. Show episodes of a specific series with a specific rating
5. Show movies with a specific rating
6. Rate a video
7. Exit
Enter your choice: 1
movieExample1 (Drama) - 3.5/5 - 1:30
movieExample2 (Mystery) - 3.5/5 - 2:20
Type y to continue:
```

4. Argument relating each project part.

4. a. The proper classes are identified

The classes identified in the UML diagram represent the key components of a streaming service's video management system. The Video class serves as the base class for general video attributes and methods, while the Movie and Series classes represent specific types of videos. The Episode class manages individual episodes of a series, and the VideoEngine handles the overall management of video collections. The Genre enumeration defines the types of genres that apply to the videos. Each class serves a specific purpose and reflects the real-world entities they represent, ensuring that the correct classes are identified.

4. b. Inheritance is implemented properly

In order to utilize inheritance in an effective manner, it is recommended that the classes representing movies and series inherit from the Video base class. This enables both subclasses to utilize common attributes and methods, such as ID, title, and getRating(), defined in the Video class. Furthermore, the Series class is associated with the Episode class, thereby demonstrating hierarchical relationships and further enhancing code reuse and organization. This implementation of inheritance facilitates the future expansion of the system to accommodate additional video types.

4. c. Access modifiers are implemented properly

Data and methods are encapsulated using the right access modifiers (e.g., public, private, protected). The use of private members prevents internal states from being accessed directly from outside the class, resulting in higher data integrity. Because of the various private members, public methods are used to control access to them. For example, the getRating() method is public so anything can call it to get the rating of a video, but it is delegated to the private member variable for a rating. It basically enforces that the internal implementation is wrapped up, which again is good for hiding and separation of "concerns."

4. d. Method overwriting is implemented properly

We use Method overriding to give a specific implementation of a base class method. The Video class has a pure virtual function getRating() which is overridden in Movie and Episode in order to return their ratings. This gives the sub-classes a chance to determine how ratings are done, assuring that the appropriate rating method is done according to each video type. This approach to method overriding complies with the Polymorphism concept and allows the system to manage different types of video accordingly.

4. e. Polymorphism is implemented properly

Polymorphism allows methods to be used interchangeably among Video objects and its subclasses. For instance, both Movie and Series objects can be part of a Video collection, enabling methods like getAverageRating() to be called on any Video object regardless of its specific subclass. In this code, the getRaiting() method is a pure virtual function in the Video class.

4. f. Abstract classes are implemented properly

The Video class is defined as an abstract class with at least one pure virtual function, getRating(). This makes Video a base class that cannot be instantiated on its own, ensuring that only concrete subclasses (Movie and Series) are instantiated. The abstract class establishes a common interface for all video types, enforcing a contract for the subclasses to implement the getRating() method. This use of abstract classes promotes a clear and consistent design, ensuring that all video types adhere to a defined structure.

4. g. At least one operator is overloaded properly

The addition operator (+) for the Series class exemplifies operator overloading. This implementation is pivotal for comprehending how to extend the functionality of built-in operators to user-defined types, thereby enhancing code readability and expressiveness. This operator enables a Series object to utilize the + operator to add an Episode object directly, thereby enhancing the syntactical elegance and intuitive nature of the code.

4.1 Why is that your best solution?

The design aligns with object-oriented principles to create a scalable, maintainable, and flexible system. The identification of appropriate classes ensures that each component has a clear role and responsibility, reflecting real-world entities accurately. Inheritance promotes code reuse and logical hierarchy, while encapsulation protects the internal state of objects. Method overriding and polymorphism enable dynamic behavior and flexible code reuse. Abstract classes enforce a consistent interface across video types, and operator overloading provides intuitive operations. Proper exception handling enhances system robustness and reliability. This comprehensive design approach ensures an efficient and user-friendly video management system for a streaming service.

5. Identification of cases that would prevent the project from functioning properly.

1. An error in the dataBase.txt which is the file that contains all the information of the videos.
2. An undiscovered exploit with the code's input system.
3. Lack of context for the user to successfully make use of the code..

6. Personal conclusion.

6. 1. Gilberto Malagamba Montejo

This project presented a unique opportunity to employ all the Object Oriented Programming concepts that I have heard about or even seen but never applied. It was a simple concept but a deep implementation, which was enjoyable considering the class is about applying what we have learned. I am content with our result and have acquired the vision of uses for these important concepts that in the future am willing to dive deeper into.

6. 2. Maria Luisa Ramos Contreras

Building this Video Management System was a good experience on how to leverage the power of Object-Oriented Programming (OOP) to build a maintainable, flexible and scalable solution. I have them defined in the correct places, and with that, I am able to gather related functionality in a scalable structure that makes my code more reusable and maintainable. Through and through, this project represents the possibility of employing OOP principles to make a system work and be easier to extend and maintain. It was a very useful process which demonstrated the relevance of careful design and fine planning in software development.

6. 3. Sara Sofía Hernández Valdez

The development of a project that integrates fundamental object-oriented principles has a significant impact on my ability to grasp and apply in the context of C++. It provides a hands-on experience that bridges the gap between theoretical knowledge and practical application. Through inheritance, I learn to create hierarchical relationships; method overriding and polymorphism teach dynamic behavior management; operator overloading introduces custom behavior definitions; and exception handling prepares them to build robust, error-tolerant programs.

This comprehensive approach equips me with the requisite skills to address the challenges encountered in the real world of programming, foster a deeper understanding of OPP programming, and laying a solid foundation for future development endeavors.

7. References

Global number of Disney+ subscribers 2024. (2024). Statista; Statista. https://www.statista.com/statistics/1095372/disney-plus-number-of-subscribers-us/

Netflix: number of subscribers worldwide 2024 . (2024). Statista; Statista. https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide/