OOP Project Design Report

Student: Siuzanna Nevar
Project Title: Movie Review Platform
Date: 03.11.2025

1. Introduction

This project is a simple movie review platform where users can leave reviews and rate movies they've watched. The system stores information about users, movies, and their reviews, and calculates an average rating for each movie based on user ratings.

The project's main purpose is to create an object-oriented program that models the operation of online review platforms. It addresses the problem of organizing user feedback and helps find popular films based on ratings and reviews.

The program's main functions include user registration, adding movies, publishing reviews and ratings, calculating and displaying the average rating, and sorting and accessing reviews and rating history per user.

2. Class Design

Class	Description
User	This is the platform’s user. User can write, update and delete reviews, also can rate movie. Contains a list of his/her reviews and ratings.
Movie	Movie’s model. Stores the title, genre, release year, rating and list of reviews for this movie. Allows you to add new reviews.
Review	A class representing a user review. Contains the review text, publication date, and the user and movie IDs to which it pertains.
Rating	Movie rating model. Stores movie and user IDs, the rating given, and the date the rating was given.
Media	Base class for media items. Stores ID, name, and genre. Movie inherits from it.

Relationships

User → Review
The user creates a review, which is stored both in the movie and on the user's account.
Movie → Review
The movie contains a list of user reviews.
User → Movie
By adding a review, the user interacts with the movie.
Rating → Movie and User
The rating connects the user with the movies and shows what rating the user gave to a particular movie.

Class	Responsibility
User	Manages user actions: adding, editing, and deleting reviews. Monitors their reviews and ratings.

Movie	Stores a list of user reviews and ratings, and also allows you to add new reviews.
Review	Stores information about the review, such as the date it was written, the user who wrote the review, and the movie about which the review was written.
Rating	Stores information about ratings given to a movie by users. Can be used to calculate the average rating.
Media	Base class providing ID, name, and genre for media items.

3. Application of OOP Principles

Encapsulation

Encapsulation is implemented through private attributes, such as *__id* in the *User* class, *__review_id* in *Review*, and *__rating_id* in *Rating*. These attributes are hidden from direct external access, protecting the data from accidental modification. Properties (@property) and methods (*write_review, update_review, delete_review, rate_movie*) are used to access and modify the data, allowing for control over the change logic and automatic updating of related values, such as the review modification date.

Inheritance

Inheritance is used in the *Movie* class, which inherits from *Media*. Common attributes such as id, name, and genre are defined in the base *Media* class, avoiding code duplication and allowing *Movie* to focus on specific data: reviews, ratings, and average rating calculation.

Polymorphism

Polymorphism manifests itself in the fact that different objects (*User* and *Movie*) interact with the same entities (*Review*, *Rating*) through unified interfaces. For example, the *add_review* method in Movie and the *write_review* method in *User* interact with *Review* objects in the same way, regardless of who created them, ensuring consistent system behavior.

Abstraction

Abstraction is implemented through the *Media* class, which defines a common interface for media objects. This allows the platform to be easily extended with other content types without changing the logic behind existing classes. Furthermore, the interface of the *User* and *Movie* classes hides the internal structures for storing reviews and ratings (dictionaries), providing high-level methods for working with them.

4. Use of Design Patterns

The project includes an element of the *Factory Method* design pattern.
The *User* class acts as a factory for creating *Review* and *Rating* objects.
Instead of creating reviews and ratings directly, the user calls methods *write_review()* and *rate_movie(),* which encapsulate the creation logic. This approach hides object creation details (ID generation, validation, timestamping) and centralizes the process, making the code easier to maintain and extend.

5. SOLID Principles

Each class in the system is responsible for only one specific task, which demonstrates SRP. The *User* class manages user actions such as creating, updating, and deleting reviews, and assigning ratings. The *Movie* class is responsible for storing reviews and ratings related to a specific film and calculating the average rating. The *Review* and *Rating* classes each store only the data associated with a user's review or rating. This clear separation of responsibilities makes the code easier to understand, maintain, and modify.

The project also follows the Open/Closed Principle (OCP). Classes are designed so they can be extended without modifying their existing code. For example, the *Media* base class allows new media types (such as a TV series or documentary) to be added simply by creating a subclass. The *Movie* class can also be expanded with additional functionality—such as sorting or filtering reviews—without altering its current logic. This ensures the system remains flexible and open for extension while staying closed to internal modification.

6. Testing Strategy

The project was tested using the *pytest* framework, which provides a simple and readable way to write unit tests. The tests covered all core functions of the system, including creating, updating, and deleting reviews, adding ratings, and recalculating a movie's average rating. Error handling was also tested, such as preventing empty review text, rejecting invalid rating values, and handling non-existent review IDs. Test scenarios included verifying that empty reviews raise a *ValueError*, updated reviews correctly change their text, multiple ratings produce the correct average, and deleted reviews are removed both from the user and the movie. Overall, the tests ensured that the system behaves correctly in normal use and reliably handles incorrect input.

7. Challenges and Design Decisions

One of the challenges was deciding how to organize the storage of reviews and ratings. A dictionary-based storage structure was chosen, with a unique ID as the key. This allowed for quick object searches, updates, and deletions without having to traverse the entire list.

There was also a question about whether to include the update date management in the *Review* class itself or in the *User* functionality. Ultimately, the logic was left within the *Review* class so that the date would be automatically updated when the text changed. This increased the object's autonomy and simplified review processing.

Furthermore, a decision was made to create a base *Media* class so that the project could be extended with other media types. This is a minor architectural complication, but it makes the project more flexible in the future.

The project demonstrates the application of OOP principles to the creation of a simple movie review platform. Its strengths include a clear division of responsibilities between classes, the use of encapsulation, the flexibility of the structure through inheritance, and ease of extensibility. With more time, it would have been possible to add review sorting (by date or length), review comments, data storage in files or a database, a full-fledged user interface, and an extended media model, including series and episodes. Working on the project allowed for a deeper understanding of OOP application architecture, the importance of structured code, and logic testing.