Midterm: MockBuster

Objective

The year is 2007, as a successful member of the DevBuild program, you have been hired to move the struggling movie rental shop, Mockbuster, into the 21st century. In order to do so, they have asked you to build a console application that digitizes their entire movie catalog.

Note: Application must compile to receive points.

Build Specifications

The application will have a minimum of four classes: Movie, Admin, User, and MoviesRepo

Movie Repository Class

This class is used to house the repository of movies in the application. This class should meet the following requirements:

1. A static method that returns a list of 5 movie objects **(10 points)**

Note: This is the repository you will be displaying to the user and utilizing the CRUD actions on. (Create, Read, Update, Delete)

Movie Class

Customers are constantly asking for recommendations and forgetting names of movies. We want all the hardcore fans of Nick Cage, Michael Bay or Akira Kurosawa to have easy access to their films, so each movie stored within your application should at minimum have the following information documented:

1. Fields/properties
    a. Movie Name **(2 points)**
    b. Main Actor **(2 points)**
    c. Genre **(2 points)**
    d. Director **(2 points)**

2. A constructor that assigns values to each of the properties **(5 points)**
3. An overridden ToString method that return a formatted string that contains Movie Name, Main Actor, Genre, and Director of the movie object **(10 points)**

User Class

The User should be used as the base class and account for a regular client of the application. A user has limited functionality such as viewing list of movies as well as filtering by the specific properties. The requirements for this class are as follows:

1. A method to filter by genre and returns a list of movies of only those genres **(10 points)**
2. A method to filter by movie name and returns a list of movies of only those movie names **(10 points)**
3. A method to filter by main actors and returns a list of movies of only those main actors **(10 points)**
4. A method to filter by director and returns a list of movies of only those directors **(10 points)**

Admin Class

The admin should be derived from the User class thus giving it the same functionality of filtering. In addition, an admin should have the ability to add, update, and delete movies from the repository. The requirements for this class are as follows:

1. A method that that takes the movie object as an argument and adds it to the movie repository **(10 points)**

Note: This functionality should not allow duplicate movies to be added. You have free choice as to where you validate duplicates.

2. The functionality to edit a movie may require multiple methods. The end result is that the user is able to retrieve the movie to be updated in the repository, choose the property to update, and have the updated movie added to the repository.
    a. Inform the user if the movie was updated and display the updated movie details **(10 points)**
    b. If there was no movie found to be updated, inform the user
3. A method to remove the admin selected movie from the movie repository
    a. Inform the user if the movie was successfully removed or if it does not exist **(10 points)**

Program Class

The Program class will act as your User Interface. This is where you will control the flow of the application, show available menus based on a client's role, and allow the client to access the functionality built out in the other classes. This class should meet the following requirements:

1. If the client is a user, they should only have access to functionality specified below
    a. Filter by movie name **(5 points)**
    b. Filter by main actor **(5 points)**
    c. Filter by genre **(5 points)**
    d. Filter by director **(5 points)**
2. Only objects that match the filter should be displayed to the console. The displayed items must contain all properties (movie name, main actor, genre, director) **(5 points)**
3. If the client is an admin, they should have access to same filter functionality of a user **(5 points)**
4. Admins also have the following functionality
    a. Ability to add a new movie to the movie repository **(5 points)**
    b. Ability to edit an existing movie in the movie repository **(5 points)**

    c. Ability to delete an existing movie in the movie repository **(5 points)**
5. Whether the client is a user or admin, the application should return them back to a menu of options until the client has opted to quit **(5 points)**

Client Validation

Client experience is essential to the overall useability of your application. Validate all client input and ensure the client receives messages for any invalid input.

1. All client input should be validated and gives client appropriate messaging to input valid data **(10 points)**

Unit Testing

As an engineer you should have pride in the quality of your code and your final product. Unit testing allows us to ensure our functionality is performing as expected. You are required to have at least one effective unit test on each of the following methods:

Note: Effective is defined as the unit test actually tests against the expected output of your method.

1. Properly set up unit test **(5 points)**
2. User Class Methods
    a. Filter By Movie Name **(10 points)**
    b. Filter By Main Actor **(10 points)**
    c. Filter By Genre **(10 points)**
    d. Filter By Director **(10 points)**
3. Admin Class Methods
    a. Add **(10 points)**
    b. Edit **(10 points)**
    c. Remove **(10 points)**
4. Repository Method
    a. Get All Movies **(10 points)**

Build Spec #	Points Possible	Points Scored
Movie Repository Class		
1	10	
Movie Class		
1a	2	
1b	2	
1c	2	
1d	2	
2	5	
3	10	
User Class		
1	10	
2	10	
3	10	
4	10	
Admin Class		
1	10	
2a	10	
2b	10	
3a	10	
Program Class		
1a	5	
1b	5	
1c	5	
1d	5	
2	5	
3	5	

4a	5	
4b	5	
4c	5	
5	5	
Client Validation		
1	10	
Unit Testing		
1	5	
2a	10	
2b	10	
2c	10	
2d	10	
3a	10	
3b	10	
3c	10	
4a	10	
Total	258	