Ministerul Educației și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

**Raport**

**la TMPS**

Tema: Un sistem de administrarea a filmelor.

**Realizat de : Catărău** Valeriu, gr. TI-204.

Chișinău – 2023

Contents

[**1.Introducere** 3](#_Toc136624117)

[**2.Etapele de creare a sistemului** 4](#_Toc136624118)

[**2.1. Funcționalitățile și cerințele acestuia** 4](#_Toc136624119)

[**2.2 Design patterns utilizate** 4](#_Toc136624120)

[**3.Implementarea** 5](#_Toc136624121)

[**Concluzie** 16](#_Toc136624122)

# **1.Introducere**

În cadrul cursului de Teorii și Mecanisme de Proiectare a Sistemelor, am avut oportunitatea de a studia diverse tipuri de design pattern-uri și principiile SOLID. Această cunoaștere ne-a oferit fundația necesară pentru a aborda proiectarea unui sistem complex. Pentru proiectul de final de an, am ales să dezvolt un sistem de gestionare a filmelor. Scopul sistemului este de a permite utilizatorilor să adauge filme, să le șteargă, să le evalueze și să le adauge într-o categorie "De urmărit". Am aplicat mai multe tipuri de design pattern-uri pentru a asigura o structură flexibilă, ușor de întreținut și extensibilă. Pentru a garanta o instanțiere unică și accesul global la sistemul de filme, am utilizat Singleton pattern. Aceasta asigură că există o singură instanță a sistemului și oferă un punct central de acces. Pentru a crea obiecte de tip film într-un mod flexibil și modular, am folosit Builder pattern. Această abordare ne-a permis să construim obiecte de tip film într-un mod pas cu pas, adăugând și setând atributelor în etape separate. Pentru a facilita interacțiunea cu sistemul de filme, am implementat un Facade pattern. Aceasta oferă o interfață simplificată prin care utilizatorii pot adăuga, șterge și evalua filme fără a fi nevoie să gestioneze detaliile complexe ale sistemului.Pentru a extinde funcționalitatea unui film într-un mod dinamic, am aplicat Decorator pattern. Acesta ne-a permis să adăugăm funcționalități suplimentare la obiectele film, cum ar fi descrierea sau etichetele speciale, fără a afecta structura de bază a clasei. Pentru a itera prin lista de filme, am implementat Iterator pattern. Acesta oferă o modalitate simplă și consistentă de a parcurge și accesa filmele din sistem.Pentru a realiza salvarea și restaurarea stării unui film, am utilizat Memento pattern. Acesta ne-a permis să capturăm starea unui film într-un obiect memento, astfel încât să putem reveni la acea stare ulterior, dacă este necesar.Prin aplicarea acestor design pattern-uri, am creat un sistem de filme. Acesta ne permite să adăugăm, să ștergem, să evaluăm și să categorisim filmele într-un mod modular și eficient. Utilizând principiile SOLID, am asigurat o arhitectură solidă și ușor de întreținut pentru sistemul nostru de filme. Proiectul de an reprezintă un exemplu practic al modului în care putem aplica conceptele teoretice în dezvoltarea unui sistem real, folosind design pattern-uri și principiile SOLID ca ghid pentru o proiectare de calitate.

# **2.Etapele de creare a sistemului**

Am analizat cerințele sistemului și am identificat design pattern-urile potrivite care să fie folositoare și să aducă beneficii în contextul specific al sistemului de filme. Am ales design pattern-uri care să ne ajute să creăm un sistem modular, extensibil și ușor de întreținut.

## **2.1. Funcționalitățile și cerințele acestuia**

Sistemul de filme pe care l-am creat îndeplinește diverse funcționalități și cerințe. Utilizatorii au posibilitatea de a adăuga filme noi în sistem, dar și de a șterge filme existente în cazul în care nu mai sunt relevante sau dorite. O caracteristică importantă a sistemului este posibilitatea de a adăuga filme în diferite categorii, precum categoria "De urmărit" sau "Will Watch". Aceasta permite utilizatorilor să-și organizeze filmele și să le gestioneze mai eficient. De exemplu, utilizatorii pot adăuga filme pe care doresc să le vizioneze în viitor în categoria "Will Watch", astfel încât să le poată găsi și accesa mai ușor atunci când este momentul potrivit. Pentru a evalua filmele și a le atribui o notă, sistemul permite utilizatorilor să adauge o notă pentru fiecare film. Aceasta oferă utilizatorilor posibilitatea de a-și exprima părerea și evaluarea despre fiecare film în parte. Evaluările pot fi folosite ulterior pentru a oferi recomandări personalizate utilizatorilor sau pentru a sorta filmele în funcție de rating-ul acordat. În concluzie, sistemul nostru de filme permite utilizatorilor să adauge și să șteargă filme, să le atribuie categorii, cum ar fi "Will Watch", și să le evalueze prin acordarea unei note. Aceste funcționalități sunt esențiale pentru o experiență completă și personalizată în cadrul sistemului de filme.

## **2.2 Design patterns utilizate**

În procesul de creare a sistemului de filme, am început prin proiectarea arhitecturii acestuia. Pentru a ne asigura că nu lipsește nimic și că avem o structură solidă, am utilizat limbajul UML pentru a concepe diagrame de clase, diagrame de secvențe și alte diagrame relevante.De exemplu, am folosit Singleton pattern pentru a asigura că există o singură instanță a sistemului de filme în cadrul aplicației noastre. Acest lucru ne-a permis să evităm crearea mai multor instanțe și să avem un punct central de acces la funcționalitățile sistemului.Pentru a facilita crearea obiectelor de tip film într-un mod flexibil, am utilizat Builder pattern. Acesta ne-a permis să construim obiectele de film pas cu pas, oferind metode separate pentru setarea atributelor și construirea finală a obiectului.Pentru a abstractiza complexitatea interacțiunii cu sistemul de filme, am aplicat Facade pattern. Acesta ne-a oferit o interfață simplificată prin intermediul căreia utilizatorii pot interacționa cu sistemul, ascunzând detaliile de implementare.De asemenea, am utilizat Decorator pattern pentru a extinde funcționalitățile unui film într-un mod flexibil. Acest pattern ne-a permis să adăugăm sau să modificăm comportamentul unui film prin învelirea acestuia în obiecte decorator suplimentare.Am aplicat și Iterator pattern pentru a itera prin lista de filme și a accesa fiecare film într-un mod consistent și simplu.În plus, am utilizat Memento pattern pentru a salva și restaura starea unui film. Acesta ne-a oferit posibilitatea de a captura starea unui film într-un obiect memento și de a reveni la acea stare ulterior, dacă este necesar.Toate aceste design pattern-uri au fost alese cu grijă pentru a se potrivi cu nevoile și cerințele sistemului nostru de filme. Acestea au contribuit la crearea unei arhitecturi modulară și ușor de extins, oferind un sistem de filme flexibil și scalabil.

# **3.Implementarea**

În continuare, veți găsi prezentată implementarea sistemului de filme, inclusiv codurile sursă și clasele corespunzătoare, în format UML. Aceasta oferă o vizualizare completă și structurată a arhitecturii sistemului nostru. Prin intermediul diagramei UML, putem înțelege mai bine interacțiunile între clase și relațiile acestora, facilitând dezvoltarea și întreținerea sistemului de filme.

Pattern-ul Singleton este un pattern de design care permite crearea unei singure instanțe a unei clase și oferă un punct de acces global la acea instanță. În acest caz, clasa **MovieSystem** are o metodă numită **getInstance** care verifică dacă instanța există deja și, în caz contrar, o creează folosind metoda **init**. Apoi, instanța este returnată.Prin utilizarea acestui pattern, putem asigura că există o singură instanță a clasei **MovieSystem** și că accesul la acea instanță se face prin apelul metodei **getInstance**. Astfel, în întregul cod, vom avea o singură instanță a **MovieSystem** și putem adăuga sau șterge filme din acea instanță prin apelul metodelor **addMovie** și **deleteMovie**.

// Singleton

const MovieSystem = (function () {

    let instance;

    function init() {

        const movies = [];

        return {

            addMovie: function (movie) {

                movies.push(movie);

            },

            deleteMovie: function (movie) {

                const index = movies.indexOf(movie);

                if (index !== -1) {

                    movies.splice(index, 1);

                }

            },

            getMovies: function () {

                return movies;

            },

        };

    }

    return {

        getInstance: function () {

            if (!instance) {

                instance = init();

            }

            return instance;

        },

    };

})();

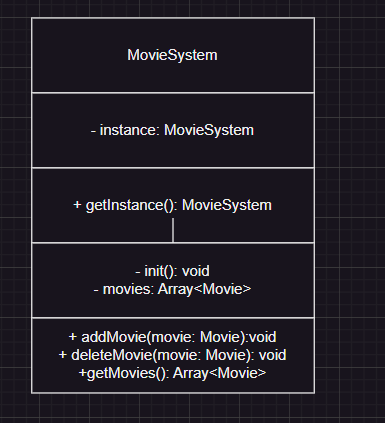


Figura 1 – Clasa uml pentru clasa MovieSystem

Pattern-ul Builder este utilizat atunci când dorim să construim obiecte complexe pas cu pas, cu opțiuni diferite disponibile în timpul construirii. În loc să avem un constructor cu o mulțime de parametri, folosim metode de setare pentru a configura diferitele proprietăți ale obiectului într-o manieră mai flexibilă.Clasa **MovieBuilder** este responsabilă de construirea unui obiect de tip film. Are câmpuri pentru **title**, **description**, **image** și **rating**. Metodele precum **setTitle**, **setDescription**, **setImage** și **setRating** sunt utilizate pentru a seta valorile acestor câmpuri și returnează instanța curentă a **MovieBuilder**, permițând astfel apeluri în lanț (chaining) pentru a configura mai multe proprietăți.Metoda **build** este utilizată pentru a finaliza procesul de construcție și returnează un obiect de tip film cu proprietățile configurate anterior în **MovieBuilder**.Prin urmare, utilizând clasa **MovieBuilder**, putem construi un obiect de tip film pas cu pas și putem configura doar proprietățile necesare în funcție de nevoile noastre. Acest pattern face codul mai lizibil și mai ușor de întreținut atunci când avem obiecte complexe cu o mulțime de proprietăți opționale.

// Builder

class MovieBuilder {

    constructor() {

        this.title = '';

        this.description = '';

        this.image = '';

        this.rating = 0;

    }

    setTitle(title) {

        this.title = title;

        return this;

    }

    setDescription(description) {

        this.description = description;

        return this;

    }

    setImage(image) {

        this.image = image;

        return this;

    }

    setRating(rating) {

        this.rating = rating;

        return this;

    }

    build() {

        return {

            title: this.title,

            description: this.description,

            image: this.image,

            rating: this.rating,

        };

    }

}

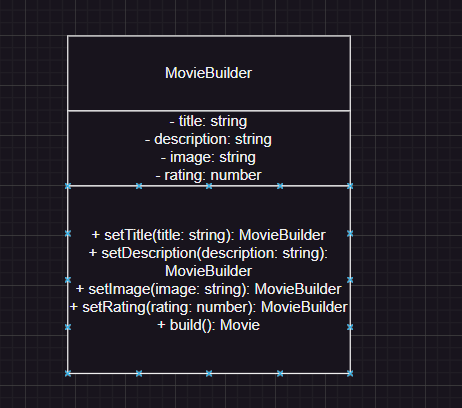


Figura 2 – Clasa uml pentru clasa MovieBuilder.

Pattern-ul Prototype este utilizat atunci când dorim să creăm noi obiecte prin clonarea unui obiect existent în loc de a le crea de la zero. Aceasta oferă o modalitate eficientă de a crea copii ale obiectelor, evitând construcția costisitoare a unui obiect nou de fiecare dată.Clasa **MoviePrototype** reprezintă prototipul unui obiect de tip film. Constructorul primește parametri precum **title**, **description**, **image** și **rating** și le atribuie membrilor corespunzători ai obiectului.Metoda **clone** este definită în clasa **MoviePrototype** și este responsabilă pentru clonarea obiectului curent. Aceasta creează un nou obiect **MoviePrototype** cu aceleași valori pentru **title**, **description**, **image** și **rating** ca obiectul original și îl returnează.Prin utilizarea metodei **clone**, putem crea rapid copii ale unui obiect **MoviePrototype**, evitând astfel crearea și inițializarea unui obiect nou de fiecare dată.Astfel, pattern-ul Prototype facilitează crearea de copii ale unui obiect prin clonarea prototipului existent, permițând astfel o creare eficientă a obiectelor similare.

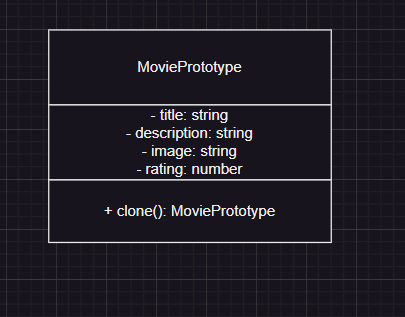


Figura 3 – Clasa uml pentru clasa Prototype.

Pattern-ul Facade este utilizat pentru a oferi o interfață simplificată către un subsistem complex de clase. Acesta ascunde detalii complicate și furnizează o interfață unificată și mai ușor de utilizat pentru utilizatorii subsistemului.Clasa **MovieFacade** acționează ca un "fațadă" către un obiect de tip film. Constructorul primește un obiect **movie** și îl stochează în membrul **this.movie**.Metoda **getFullInfo()** oferă o interfață simplificată pentru a obține informații complete despre filmul dat. Aceasta returnează un șir de caractere care conține titlul filmului (**this.movie.title**), descrierea filmului (**this.movie.description**) și ratingul filmului (**this.movie.rating**) formatat cu o precizie de 1 zecimală (**toFixed(1)**).Prin intermediul clasei **MovieFacade**, utilizatorii nu trebuie să cunoască detaliile interne ale obiectului **movie**. Aceștia pot accesa informațiile complete despre film printr-o singură metodă, fără a fi nevoie să interacționeze direct cu obiectul **movie** sau cu alte clase asociate.În concluzie, pattern-ul Facade este utilizat aici pentru a oferi o interfață simplificată către obiectul **movie**, ascunzând detaliile complexe și oferind un acces ușor la informațiile complete despre film.

// Facade

class MovieFacade {

    constructor(movie) {

        this.movie = movie;

    }

    getFullInfo() {

        return `Title: ${this.movie.title}\nDescription: ${this.movie.description}\nRating: ${this.movie.rating.toFixed(1)}`;

    }

}

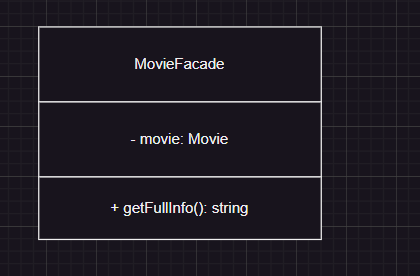


Figura 4 – Clasa uml pentru clasa MovieFacade.

Pattern-ul Decorator este utilizat pentru a extinde funcționalitatea unei clase existente prin adăugarea de comportamente suplimentare la timpul de execuție, fără a modifica structura de bază a clasei.Avem două clase care folosesc pattern-ul Decorator: **MovieDecorator** și **MovieRatingDecorator**.Clasa **MovieDecorator** este clasa de bază a decoratoarelor de film. Primește un obiect **movie** în constructor și îl stochează în membrul **this.movie**. Metoda **getFullInfo()** utilizează obiectul **movie** pentru a obține informații complete despre film.Clasa **MovieRatingDecorator** extinde clasa **MovieDecorator** și adaugă funcționalitatea de rating la informațiile complete despre film. Primește, în plus față de **movie**, și un parametru **rating** în constructor și îl stochează în membrul **this.rating**. Suprascrie metoda **getFullInfo()** pentru a adăuga ratingul utilizatorului la informațiile complete obținute din **super.getFullInfo()**. Astfel, rezultatul returnat de **getFullInfo()** include și ratingul utilizatorului.Prin intermediul acestor clase decorator, putem extinde funcționalitatea obiectelor de tip film fără a le modifica direct. Putem adăuga decoratori suplimentari pentru a adăuga și alte comportamente suplimentare la obiectele **MovieDecorator**.

Astfel, pattern-ul Decorator permite adăugarea de comportamente suplimentare la obiectele existente, fără a le afecta structura de bază și fără a crea clase derivate suplimentare. // Decorator

class MovieDecorator {

    constructor(movie) {

        this.movie = movie;

    }

    getFullInfo() {

        return this.movie.getFullInfo();

    }

}

class MovieRatingDecorator extends MovieDecorator {

    constructor(movie, rating) {

        super(movie);

        this.rating = rating;

    }

    getFullInfo() {

        return `${super.getFullInfo()}\nUser Rating: ${this.rating.toFixed(1)}`;

    }

}

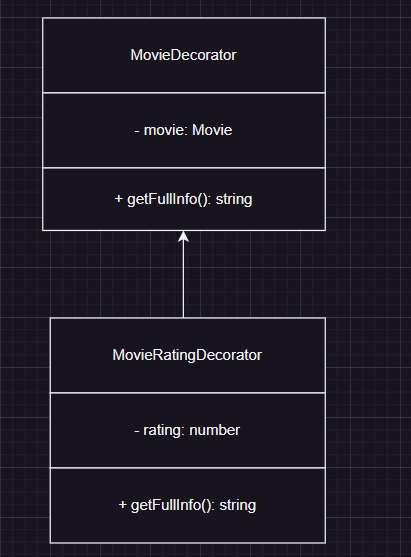


Figura 5 – Relatia intre clasa MovieDecorator si MovieRatingDecorator.

Pattern-ul Iterator oferă o modalitate de a itera printr-o colecție de obiecte într-un mod secvențial, fără a expune detaliile de implementare ale colecției respective.Clasa **MovieIterator** acționează ca un iterator pentru o colecție de filme. Constructorul primește un parametru **movies**, care reprezintă colecția de filme pe care dorim să o iterăm. De asemenea, inițializează membrul **index** cu valoarea 0 pentru a urmări poziția curentă în colecție.Metoda **hasNext()** verifică dacă există un următor element disponibil în colecție. Aceasta returnează **true** dacă **index** este mai mic decât lungimea colecției de filme, indicând că mai există elemente de iterat, și **false** în caz contrar.Metoda **next()** returnează următorul element din colecție și avansează **index** cu o poziție pentru a indica următorul element din colecție.Prin utilizarea clasei **MovieIterator**, putem itera prin colecția de filme într-un mod secvențial, fără a fi nevoie să accesăm direct colecția de filme sau să cunoaștem detalii despre implementarea acesteia.Astfel, pattern-ul Iterator facilitează iterarea prin colecții de obiecte într-un mod abstractizat, oferind o interfață simplă și consistentă pentru a accesa elementele colecției.

// Iterator

class MovieIterator {

    constructor(movies) {

        this.movies = movies;

        this.index = 0;

    }

    hasNext() {

        return this.index < this.movies.length;

    }

    next() {

        return this.movies[this.index++];

    }

}

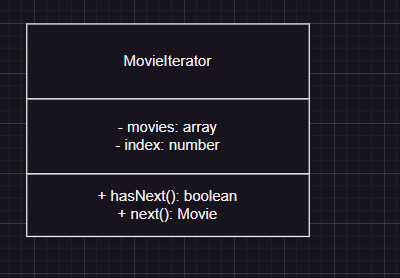


Figura 6 – Clasa uml pentru clasa MovieIterator.

**Interface:**

function displayMovies(movies) {

    const moviesDiv = document.getElementById("movies");

    moviesDiv.innerHTML = "";

    movies.forEach((movie) => {

        const movieContainer = document.createElement("div");

        movieContainer.className = "movie-container";

        const movieImage = document.createElement("img");

        movieImage.className = "movie-image";

        movieImage.src = movie.image;

        const movieDetails = document.createElement("div");

        movieDetails.className = "movie-details";

        const movieTitle = document.createElement("h2");

        movieTitle.className = "movie-title";

        movieTitle.textContent = movie.title;

        const movieDescription = document.createElement("p");

        movieDescription.className = "movie-description";

        movieDescription.textContent = movie.description;

        const movieRating = document.createElement("div");

        movieRating.className = "movie-rating";

        movieRating.textContent = `Rating: ${movie.rating.toFixed(1)}`;

        const rateButton = document.createElement("button");

        rateButton.textContent = "Rate Movie";

        rateButton.addEventListener("click", () => showRatingForm(movie));

        movieDetails.appendChild(movieTitle);

        movieDetails.appendChild(movieDescription);

        movieDetails.appendChild(movieRating);

        movieDetails.appendChild(rateButton);

        movieContainer.appendChild(movieImage);

        movieContainer.appendChild(movieDetails);

        moviesDiv.appendChild(movieContainer);

    });

}

function showRatingForm(movie) {

    const ratingForm = document.getElementById("rating-form");

    const ratingInput = document.getElementById("rating-input");

    const submitButton = document.getElementById("submit-rating");

    ratingForm.style.display = "block";

    ratingInput.value = "";

    submitButton.addEventListener("click", () => {

        const rating = parseFloat(ratingInput.value);

        if (!isNaN(rating) && rating >= 0 && rating <= 10) {

            addRating(movie, rating);

            ratingForm.style.display = "none";

        } else {

            alert("Invalid rating. Please enter a number between 0 and 10.");

        }

    });

}

Funcția **displayMovies** primește o listă de filme (**movies**) ca argument și are rolul de a afișa aceste filme într-un format specific într-un element HTML cu id-ul "movies". În interiorul funcției, se accesează elementul HTML cu id-ul "movies" folosind **document.getElementById("movies")**, iar apoi se setează conținutul acestui element la un șir vid (**""**) prin intermediul proprietății **innerHTML**. Acest lucru are ca efect ștergerea oricărui conținut existent în acel element.Urmează un bloc **forEach** care iterează prin fiecare film din lista **movies**. Pentru fiecare film, se creează elemente HTML corespunzătoare, cum ar fi un container pentru film (**movieContainer**), o imagine pentru film (**movieImage**), detalii despre film (**movieDetails**), titlul filmului (**movieTitle**), descrierea filmului (**movieDescription**), ratingul filmului (**movieRating**), și un buton pentru a evalua filmul (**rateButton**). Aceste elemente sunt create folosind metoda **createElement** și li se atribuie clase CSS și conținut adecvat prin intermediul proprietăților **className** și **textContent**. De asemenea, un eveniment **click** este adăugat butonului pentru a apela funcția **showRatingForm** și a afișa formularul de evaluare a filmului.Ulterior, elementele create sunt atașate la elementul principal "moviesDiv" prin intermediul metodelor **appendChild**. Astfel, structura HTML a fiecărui film este construită și atașată la containerul principal "moviesDiv". Această secvență de cod creează o reprezentare vizuală a fiecărui film în interiorul elementului HTML cu id-ul "movies".Funcția **showRatingForm** primește un obiect **movie** ca argument și are rolul de a afișa un formular de evaluare a filmului pentru filmul dat. În interiorul funcției, se accesează elemente HTML relevante prin intermediul **document.getElementById**. Elementul formularului (**ratingForm**), elementul de intrare pentru rating (**ratingInput**), și butonul de submit (**submitButton**) sunt selectate pe baza id-urilor lor.După aceea, stilul elementului formularului (**ratingForm**) este setat la "block" pentru a fi afișat în pagina web. Valoarea elementului de intrare a ratingului (**ratingInput**) este setată la șir vid (**""**).Un eveniment **click** este adăugat butonului de submit (**submitButton**). În interiorul acestui eveniment, se preia valoarea introdusă în elementul de intrare a ratingului (**ratingInput**) și se verifică dacă aceasta este un număr valid între 0 și 10. Dacă ratingul este valid, se apelează funcția **addRating** cu filmul și ratingul corespunzător ca argumente, și apoi se ascunde formularul de evaluare a filmului (**ratingForm**). În caz contrar, se afișează un mesaj de alertă pentru a informa utilizatorul că a introdus un rating nevalid.Această secvență de cod combină funcționalități de afișare a filmelor și de gestionare a formularului de evaluare a filmului, permițând utilizatorilor să vadă filme și să adauge ratinguri.

function addMovie(name, description, rating) {

    const movie = new MovieBuilder()

        .setTitle(name)

        .setDescription(description)

        .setRating(rating)

        .build();

    movieSystem.addMovie(movie);

    displayMovies(movieSystem.getMovies());

}

Secvența de cod prezentată definește o funcție numită **addMovie** care primește trei argumente: **name** (numele filmului), **description** (descrierea filmului) și **rating** (evaluarea filmului). În interiorul funcției, se creează o nouă instanță a clasei **MovieBuilder** utilizând **new MovieBuilder()**. Apoi, pe această instanță se apelează metodele **setTitle**, **setDescription** și **setRating**, în lanț, pentru a seta valorile corespunzătoare ale proprietăților filmului în construcție. Apelul metodei **build()** pe instanța **MovieBuilder** returnează obiectul final al filmului, care a fost configurat cu titlu, descriere și evaluare.Apoi, obiectul filmului este adăugat la sistemul de filme utilizând metoda **addMovie** a instanței **movieSystem**.În cele din urmă, se afișează lista actualizată a filmelor din sistem utilizând funcția **displayMovies**.

# **Concluzie**

În concluzie, pe parcursul cursului de Teorie și Mecanisme de Proiectare a Sistemelor, am dobândit o înțelegere profundă a design pattern-urilor și am învățat cum să le aplicăm în practică. Design pattern-urile ne-au oferit un set de soluții și abordări bine definite pentru a rezolva probleme comune în dezvoltarea software-ului. Utilizând aceste concepte, am creat un sistem de filme care beneficiază de principiile solid și de avantajele oferite de design pattern-uri precum Singleton, Builder, Decorator, Abstract Factory, Iterator și Memento.Proiectul nostru de sistem de filme a profitat de modularitatea și flexibilitatea aduse de design pattern-uri, permițându-ne să adăugăm și să ștergem filme, să le evaluăm și să le adăugăm în diverse categorii, cum ar fi "will watch". De asemenea, am putut extinde funcționalitățile filmelor folosind decoratorul și am implementat salvarea și restaurarea stării cu ajutorul Memento pattern-ului. Această experiență ne-a ajutat să înțelegem beneficiile design pattern-urilor în dezvoltarea software-ului și ne-a pregătit pentru a aborda proiecte mai complexe și mai scalabile în viitor.