Movie Explorer Documentation

Introduction

Movie Explorer

The **Movie Explorer** is a dynamic and user-friendly application designed to enhance the movie-watching experience by offering movie recommendations. In an era where the vast array of available movies can be overwhelming, Movie Explorer steps in as a reliable guide, helping users discover films tailored to their preferences.

Project Overview

- **Project Title:** Movie Explorer
- **Author:** Basel Al-Raoush

Purpose

The primary goal of Movie Explorer is to simplify the process of finding movies that align with the user's taste. By leveraging a diverse database of movies, whether users are seeking random movie suggestions, exploring top-rated films by year, or narrowing down choices by genre, Movie Explorer provides a seamless and enjoyable experience.

Features

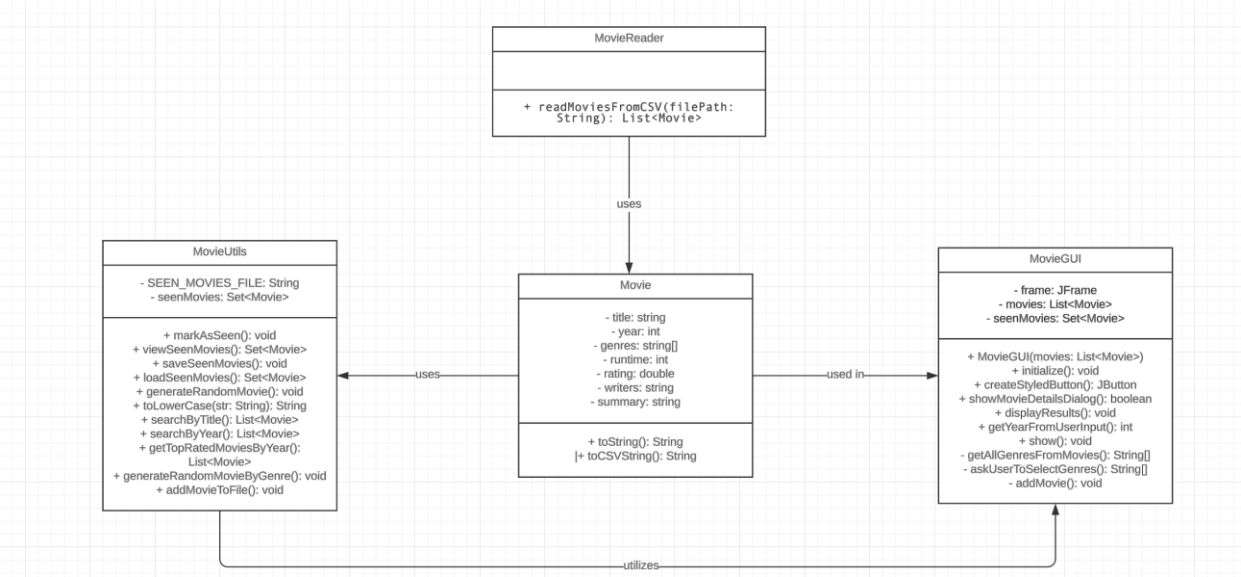
- **Random Movie Generator:** Instantly receive suggestions for a random movie selection.
- **Top Rated Movies by Year:** Explore the highest-rated movies for a specific year.
- **Genre-based Recommendations:** Find movies based on preferred genres.
- **Search Functionality:** Easily search for movies by title or release year.
- **User-friendly Interface:** The intuitive GUI ensures a smooth and enjoyable user experience.

Technology Stack

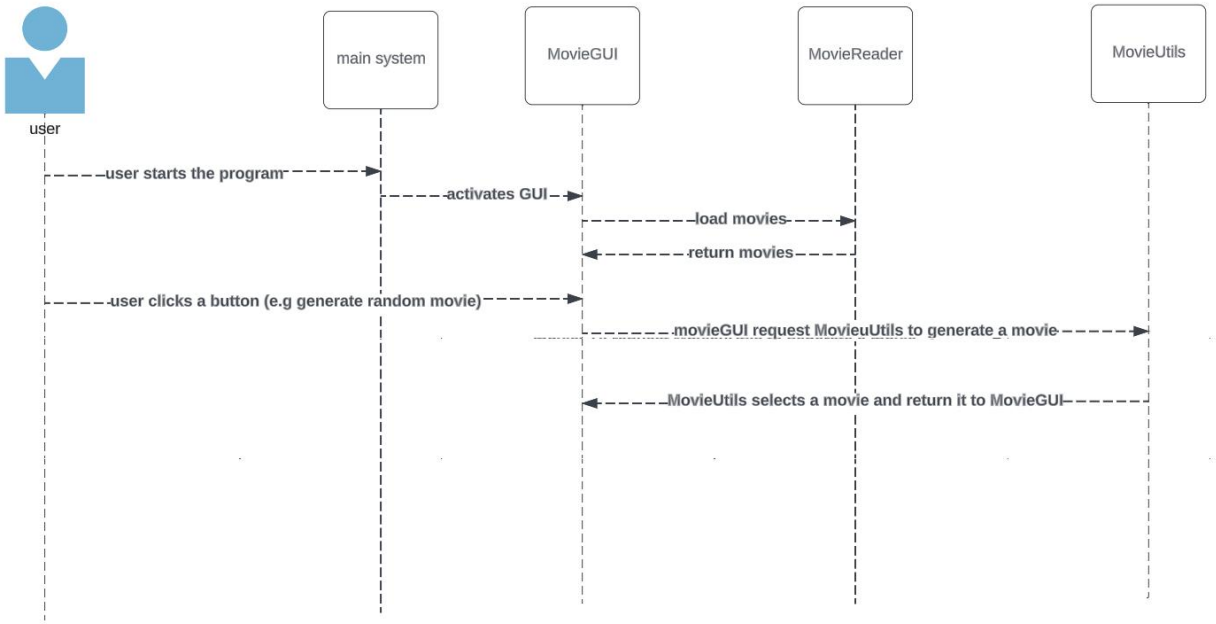
- **Programming Language:** Java
- **Graphical User Interface (GUI) Library:** Swing (Java)
- **Data Serialization:** Java Object Serialization

Architecture:

Class diagram:



Sequence diagram:



Functionality:

- **Generate Random Movie:**
  - Clicking on the "Generate Random Movie" button will display details of a randomly selected movie.
  - Users can choose to mark the movie as seen.
- **Top Rated Movies by Year:**
  - Displays the top-rated movies for a specified year.
  - Users can input the year and view the top movies.
- **Random Movie by Genre:**
  - Allows users to select genres and generates a random movie based on the chosen genres.
- **Search Movies:**
  - Provides options to search for movies by title or year.
  - Users can input search criteria and view the search results.
- **Add Movie:**
  - Enables users to add a new movie to the collection.
  - Prompts users to enter details such as title, year, genres, etc.
- **View Seen Movies:**
  - Displays a list of movies marked as seen by the user.

How to Use:

- **Launching the Application:**
  - Run the **Main** class to launch the Movie Explorer application.
- **Interacting with the GUI:**
  - The main GUI provides buttons for different functionalities.
  - Click on a button to perform the corresponding action.
- **Adding a Movie:**
  - Click on the "Add Movie" button to add a new movie to the collection.
  - Follow the prompts to enter movie details.
- **Viewing Seen Movies:**
  - Click on the "View Seen Movies" button to see a list of movies marked as seen.
- **Generating Random Movies:**
  - Use the "Generate Random Movie" button to see details of a randomly selected movie.
- **Searching for Movies:**
  - Click on the "Search Movies" button to search for movies by title or year.

Project Structure:

- **Movie Class:**
  - Represents a movie with attributes such as title, year, genres, etc.
- **MovieUtils Class:**
  - Provides utility methods for movie-related operations (e.g., generating random movies, searching).
- **MovieGUI Class:**
  - Implements the graphical user interface for the Movie Explorer application.
- **MovieReader Class:**
  - Reads movies from a CSV file and populates the movie collection.

Code Documentation

- Each class is documented with comments explaining the purpose of the class, its methods, and how to use them.
- Follows standard Java naming conventions for classes, methods, and variables.
- Using meaningful and descriptive comments within the code for the logic.

Future Enhancements

- Using advanced algorithms to personalize the recommendation depending on the user interaction with the program and their preferences.

Test cases description:

MovieGUITest

1. **createMovieGUI_ValidMovieList_ShouldNotBeNull:**
    - **Description:** Verifies that a MovieGUI instance is successfully created when provided with a valid list of movies.
    - **Test Steps:**
        - Create a sample list of movies.
        - Initialize a MovieGUI instance.
    - **Expected Result:** The MovieGUI instance should not be null.
2. **showMovieDetailsDialog_ValidMovie_ShouldReturnTrue:**
    - **Description:** Ensures that the showMovieDetailsDialog method returns true when presented with a valid movie.
    - **Test Steps:**
        - Create a sample movie.
        - Create a JFrame for the dialog.
        - Call showMovieDetailsDialog with the sample movie.
    - **Expected Result:** The method should return true, indicating that the movie details were successfully displayed.
3. **getYearFromUserInput_UserSelectsYear_ShouldReturnSelectedYear:**
    - **Description:** Checks if the getYearFromUserInput method returns the selected year when the user interacts with the dialog.
    - **Test Steps:**
        - Create a MovieGUI instance.
        - Mock user input by setting the selected year.
        - Call the getYearFromUserInput method.
    - **Expected Result:** The method should return the selected year.

MovieReaderTest

1. **readMoviesFromCSV_validFile_shouldReturnListOfMovies:**
    - **Description:** Validates that reading movies from a valid CSV file results in a non-empty list of movies.
    - **Test Steps:**
        - Read movies from a valid CSV file.
    - **Expected Result:** The list of movies should not be null or empty, and it should contain specific movies, such as "Hacksaw Ridge."
2. **readMoviesFromCSV_invalidFile_shouldThrowIOException:**
    - **Description:** Ensures that reading from an invalid CSV file throws an IOException.
    - **Test Steps:**
        - Attempt to read movies from an invalid CSV file.
    - **Expected Result:** An IOException should be thrown.
3. **readMoviesFromCSV_nonexistentFile_shouldThrowIOException:**
    - **Description:** Verifies that reading from a nonexistent file throws an IOException.
    - **Test Steps:**
        - Attempt to read movies from a nonexistent file.
    - **Expected Result:** An IOException should be thrown.

MovieUtilsTest

1. **generateRandomMovie:**
    - **Description:** Tests the generation of a random movie and ensures it is marked as seen.
    - **Test Steps:**
        - Generate a random movie using MovieUtils.
    - **Expected Result:** The set of seen movies should not be empty.
2. **generateRandomMovieByGenre:**
    - **Description:** Checks the generation of a random movie by genre and ensures it is marked as seen.
    - **Test Steps:**
        - Generate a random movie by genre using MovieUtils.
    - **Expected Result:** The set of seen movies should not be empty.
3. **addMovieToFile:**
    - **Description:** Validates the addition of a movie to a file and confirms its presence in the file.
    - **Test Steps:**
        - Add a sample movie to a temporary file.
    - **Expected Result:** The movie details should be present in the file content.
4. **markAsSeen:**
    - **Description:** Tests the marking of a movie as seen and checks its presence in the set of seen movies.
    - **Test Steps:**
        - Mark a sample movie as seen.
    - **Expected Result:** The set of seen movies should contain the marked movie.