Movie Bank System

**Grade settings**: Maximum grade: 100  
**Run**: Yes **Evaluate**: Yes  
**Automatic grade**: Yes

[***Click here to download the code template***](https://cognizant.tekstac.com/pluginfile.php/74072/mod_vpl/intro/MovieBank.zip)

***Movie Bank System***is an automated application for manipulating the various Movies seen by the customers.

They have developed an application for taking various reports based on the Movie. The details of the various functions supported by the system are provided in this case study.

You are required to write Junit test case and check the correctness of the application developed.

**Functional Requirements:**

The application has the below classes and methods implemented.

You are provided with a model class MovieDetails

**Component Specification:**MovieDetails**(Model Class)**

|  |  |  |
| --- | --- | --- |
| **Type(Class)** | **Attributes** | **Methods** |
| MovieDetails | String movieCode  String movieName  String actorName  String directorName  String genre  int yearOfRelease | Necessary getters and setters are provided.  A constructor is also provided. |

Here the genre can take a value either “Historical“ or “Crime-Thriller” or “Comedy” or “Action”.

[Note: Values are case insensitive].

**Component Specification:**InvalidMovieDetailsException **(This class inherits the Exception Class)**

|  |  |
| --- | --- |
| **Type(Class)** | **Methods** |
| InvalidMovieDetailsException | Provided with a single argument constructor – InvalidMovieDetailsException(String message) |

You are also provided with a utility class MovieBank and the below business requirements are implemented in it for which JUnit test cases are to be written and tested.

You are also provided with an utility class MovieBank with business methods.

**Component Specification:**MovieBank**(Utility Class)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Component Name** | **Type(Class)** | **Attributes** | **Methods** |
| MovieBank | MovieBank | List<MovieDetails> movieDetailsList | Getter and setter for the movieDetailsList are provided. |

The below are the requirements  implemented in the Utility class for which JUnit test cases are to be written and tested.

**Component Specification:**MovieBank**(Utility Class)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Component Name** | **Type (Class)** | **Methods** | **Responsibilities** | **Exception** |
| Validating the Genre | MovieBank | public boolean validateGenre(String genre) | Validate the Genre.  If valid return true , else this method should throw a user defined exception | Throw a user defined exception “InvalidMovieDetailsException”  if the Genre Is not “Historical” or “Crime-Thriller” or “Comedy” or “Action”. |
| View Movie Details by Movie name | MovieBank | public MovieDetails viewMovieDetailsByMovieName(String movieName) | This method should return the MovieDetails object with the movieName passed as parameter from movieDetailsList.  If the movieDetailsList  is empty or if there is no Movie details  with  the given Movie name it should throw a user defined exception | Throw a user defined exception “InvalidMovieDetailsException” if the movieDetailsList is empty or if there is  no Movie Details  in the given Movie name. |
| View the list of Movie details for a given Director name | MovieBank | public List<MovieDetails> viewMovieDetailsByDirectorName(String directorName) | This method takes the directorName as argument. It should return the list of MovieDetails for the given Director Name. If the movieDetailsList  is empty it should throw a user defined exception. | Throw a user defined exception “InvalidMovieDetailsException” if the movieDetailsList is empty. |
| View Movie details based on the year of release | MovieBank | public Map<Integer,List<MovieDetails>> viewMovieDetailsReleasedYearWise() | This method should return the  MovieDetails  objects based on the year of Release from the movieDetailsList. It returns a Map with key as number of Movies and value as movie details.  If the movieDetailsList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidMovieDetailsException” if the movieDetailsList is empty. |
| View the number of Movies for each Genre | MovieBank | public Map<String,Integer> countTotalMovieForEachGenre() | This method should return the number of MovieDetails  objects for each Genre based on the movieDetailsList. It returns a Map with key as Genre and value as count of movies.  If the movieDetailsList is empty it should throw a user defined exception. | Throw a user defined exception “InvalidMovieDetailsException” if the movieDetailsList is empty. |

You need to write Junit test for the MovieBank class.

**Testing Scenarios:**

You are provided with a class “MovieBankTest” to do this testing.

**Note:**

To perform testing the movieDetailsList should contain objects of MovieDetails.

To do this, in MovieBankTest class you are provided with a setup method. Use this method to initialize the  movieDetailsList  attribute in MovieDetails class.

Create few objects for MovieBank, populate a list with these objects and set the movieDetailsList to this list using the setMovieDetailsList method in MovieDetails class.

The below are the test methods to be implemented in MovieBankTest class.

|  |  |
| --- | --- |
| **Test Method** | **Scenarios / Responsibilities** |
| test11ValidateGenreWhenHistorical | This method should test the validateGenre method when “Historical” is passed as parameter |
| test12ValidateGenreWhenCrimeThriller | This method should test the validateGenre method when “Crime-Thriller” is passed as parameter. |
| test13ValidateGenreWhenComedy | This method should test the validateGenre method when “Comedy” is passed as parameter. |
| test14ValidateGenreWhenAction | This method should test the validateGenre method when “Action” is passed as parameter. |
| test15ValidateGenreWhenInvalid | This method should test the validateGenre method when invalid value is passed as parameter  validateGenre is expected  to throw InvalidMovieDetailsException when type is invalid.  Write JUnit to test for the exception thrown either by using appropriate annotation or by using try catch block. |
| test16ViewMovieDetailsByMovieNameWhenValid | This method should test the correctness of viewMovieDetailsByMovieName method.  Perform testing for the correctness of the value returned. |
| test17ViewMovieDetailsByMovieNameWhenInvalid | This method should test the correctness of viewMovieDetailsByMovieName method for a non existing movie name.  Perform testing for the correctness of the value returned.  viewMovieDetailsByMovieName method is expected  to throw InvalidMovieDetailsException when connection id does not exist.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test18ViewMovieDetailsByDirectorName | This method should test the correctness of  viewMovieDetailsByDirectorName method.  Perform testing for the correctness of the value returned. |
| test19ViewMovieDetailsReleasedYearWise | This method should test the correctness of viewMovieDetailsReleasedYearWise method.  Perform testing for the correctness of the value returned. |
| test20CountTotalMovieForEachGenre | This method should test the correctness of countTotalMovieForEachGenre method.  Perform testing for the correctness of the value returned. |
| test21ViewMovieDetailsByDirectorNameForEmptyList | This method should test the correctness of viewMovieDetailsByDirectorName method for an empty movieDetailsList.  viewMovieDetailsByDirectorName method is expected  to throw InvalidMovieDetailsException when movieDetailsList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test22ViewMovieDetailsReleasedYearWiseForEmptyList | This method should test the correctness of viewMovieDetailsReleasedYearWise method for an empty movieDetailsList.  viewMovieDetailsReleasedYearWise method is expected  to throw InvalidMovieDetailsException when movieDetailsList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |
| test23CountTotalMovieForEachGenreForEmptyList | This method should test the correctness of countTotalMovieForEachGenre method for an empty movieDetailsList.  countTotalMovieForEachGenre method is expected  to throw InvalidMovieDetailsException when movieDetailsList is empty.  Write JUnit to test for the exception thrown  either by using appropriate annotation or by using try catch block |

Implement the test methods and provide the needed annotation to all the methods in MovieBankTest  class.

Also this class is provided with the annotation, so that the test methods are executed in ascending order of the test method names.

You are provided with a Main class with the main method to check the correctness of the test methods written in MovieBankTest class.

Having completed writing the test methods, uncomment the code in Main class and execute the main method.