1 Purpose

The purpose of this assignment is to allow you to practice Exception Handling and File I/O as well as other object-oriented concepts covered previously.

2 Introduction

As a part of his job at a movie theater in the 90's, Mr. Filmbuff created a number of text files, each storing information about movies released in the same year.

Each text file contains zero or more lines separated by a newline character. Each line represents a movie record, specifying information about 10 movie features, namely: year, title, duration, genres, rating, score, director, actor 1, actor 2, and actor 3.

With a movie catalog containing several hundred of entries, Mr. Filmbuff finds it a bit tedious to read through his densely formatted text files and a bit frustrating when he finds typos in his catalog. Turning to you for help, he would like you to

(1) partition all valid movie records into new genre-based[1] text files,

(2) load an array of movie records from each of the partitioned text file, serializing the resulting movie array to a binary file, and

(3) deserialize (reconstruct) the serialized arrays from the binary files into a 2D-array of movie record objects, and finally provide an interactive program that allows the user to navigate a movie array, displaying user-specified number of movie-records.

Given that you are now a Java guru, this should be a piece of cake.

[1] musical, comedy, animation, adventure, drama, crime, biography, horror, action, documentary, fantasy, mystery, sci-fi, family, western, romance, and thriller

3 Your Assignment

1. Write a `Movie` class that `implements Serializable` and includes 10 instance variables, one for each feature specified in a movie record.

   Equip your `Movie` class with a pair of accessor and mutator methods for each instance variable, and override the `equals()` and `toString()` methods.

   Feel free to introduce your own variables and methods to the class in order to facilitate the the operations in this assignment, making sure that the code you introduce conforms to the important design principle of information hiding.

2. Write a `main()` method implementing the requirements of this assignment in three sequentially dependent parts described in the following pages.

```
1  public static void main(String[] args)
2  {
3    // part 1's manifest file
4     String part1_manifest = "part1_manifest.txt";
5
6    // part 2's manifest file
7     String part2_manifest = do_part1(part1_manifest /*, ... */);  // partition
8
9    // part 3's manifest file
10    String part3_manifest = do_part2(part2_manifest /*, ... */);  // serialize
11                            do_part3(part3_manifest /*, ... */);  // deserialize
12                                                                  // and navigate
13   return;
14 }
```

   A manifest file stores important information about the files, resources, and components needed to successfully build and run an application.

   The variable `part1_manifest` on line 4 represents the name of a text file that stores the names of Mr. Filmbuff's movie files.

   The call `do_part1(...)` on line 7 produces the name of the manifest file for part 2.

   Similarly, the call `do_part2(...)` on line 10 produces the name of the manifest file to be used as input in the call `do_part3(...)` on line 11.

   The comments `/*, ... */` indicate where you can optionally introduce additional arguments of your own to facilitate the tasks involved.

4 CSV-Formatted Input Files

This assignment requires input data from one or more text files, including a manifest file named `part1_manifest.txt` that stores the names of the required input files, one name per line.

The files named in `part1_manifest` may be any valid file name, and the file names listed in it each may or may not exist, and if they do exist, they may or may not be empty. If a file does not exist, you will simply record and display an error message and then move on to the next input file, if any.

The lines of an input movie file are separated by a new line character, with each line representing a movie record of ten fields:

`year`, `title`, `duration`, `genres`, `rating`, `score`, `director`, `actor 1`, `actor 2`, `and actor 3`.

The data fields themselves are separated by a comma character, a textual format called "CSV" (comma separated values). Thus, the data fields in a valid CSV-formatted record of n data fields are separated by exactly $n-1$ commas. However, should a data field contain one or more commas as part of its own data, then that field must be enclosed in "double quotes". A field without a comma may or may not be enclosed in double quotes.

For simplicity, you may assume that

- The blank spaces around the commas, the field separators, are not part of any data fields and should be ignored.
- A quoted data field begins and ends in double quotes, and it itself may not contain double quotes within its text.
- An unquoted data field does not begin with a double quote and may contain any character including double quotes.

A CSV record is valid if it has exactly 10 data fields; otherwise, it has **syntax errors**: missing or excess field(s), or missing the end quote in a quoted field.

To represent a valid movie, a CSV record must be free of syntax errors first and then free of **semantic errors**, which occur when one or more of the data fields contain invalid values:

- A valid `year` is an integer from 1990 through 1999.
- A valid `duration` is an integer from 30 through 300 minutes.
- A valid `score` is a positive `double` value less than or equal 10.

part1_manifest.txt

```
movies1990.csv
movies1991.csv
movies1992.csv
movies1993.csv
movies1994.csv
movies1995.csv
movies1996.csv
movies1997.csv
movies1998.csv
movies1999.csv
```

Syntax Errors

```
missing quotes
excess field(s)
missing field(s)
```

Semantic Errors

```
missing title
missing name(s)
missing year
invalid year
missing duration
invalid duration
missing genre
invalid genre
missing rating
invalid rating
missing score
invalid score
```

- The valid ratings are PG, Unrated, G, R, PG-13, NC-17,
- The valid genre are musical, comedy, animation, adventure, drama, crime, biography, horror, action, documentary, fantasy, mystery, sci-fi, family, romance, thriller, western

4.1 An Example of a Valid Record

```
2004,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

4.2 Examples of Records With Syntax Error

missing quotes

```
2004,"I, Robot ,115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

excess field(s)

```
2004, I, Robot ,115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing field(s)

```
2004,"I, Robot ",   Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

4.3 Examples of Records With Semantic Error

invalid year

```
-2004,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

```
MMIV,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing year

```
,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

invalid duration

```
2004,"I, Robot ",444,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

```
2004,"I, Robot ",1 hour,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing duration

```
2004,"I, Robot ",   ,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

invalid genre

```
2004,"I, Robot ",115,Aktion,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing genre

```
2004,"I, Robot ",115,Action,     ,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

invalid rating

```
2004,"I, Robot ",115,Action,PG-31,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing rating

```
2004,"I, Robot ",115,Action,     ,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

invalid score

```
2004,"I, Robot ",115,Action,PG-13,71,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

```
2004,"I, Robot ",115,Action,PG-13,ten,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing score

```
2004,"I, Robot ",115,Action,PG-13,   ,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing name(s)

```
2004,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,   ,Chi McBride
```

missing title

```
2004,   ,115,Action,PG-13,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

A syntactically valid movie record may have zero or more semantic errors:

missing title; invalid genre; missing rating

```
2004,   ,115, komedy,  ,7.1,Alex Proyas,Will Smith,Bruce Greenwood,Chi McBride
```

missing year; missing name(s)

```
,"I, Robot ",115,Action,PG-13,7.1,Alex Proyas,Will Smith,   ,Chi McBride
```

And so on.

5 Part 1: Partition Movies to Create Input Files for Part 2

Write a method `do_part1()` that partitions movie records in all the input files into CSV files, each storing valid movie records from the same genre. The method should take as a parameter the name of a manifest file storing the names of the input files supplied by Mr. Filmbuff.

The method will create a total of 19 text files:

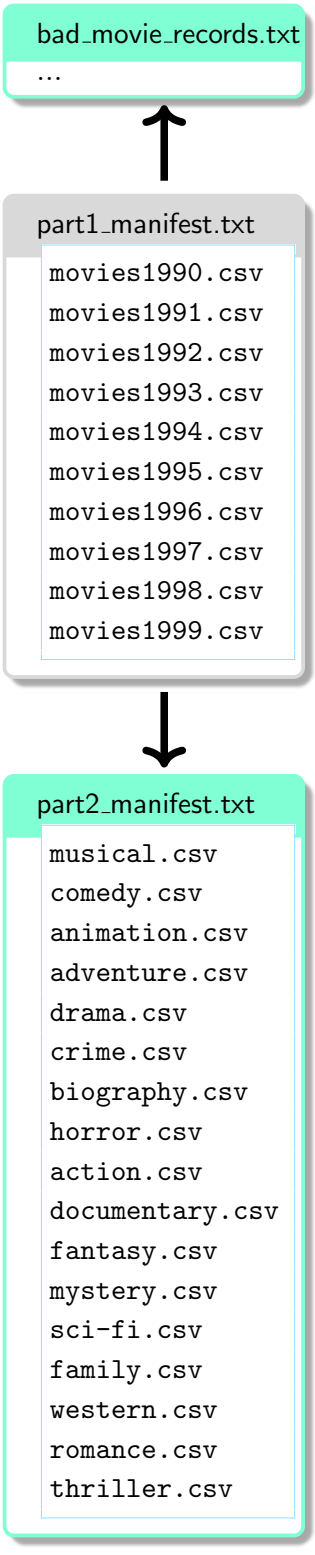
- A file named `bad_movie_records.txt` storing the movie records with syntax or semantic errors (i.e., the invalid records.)
- 17 CSV files, each storing valid movie records in a specific genre, to be used as input in Part 2. The file names are created by appending the text `".csv"` to each of the 17 possible genres, as shown in the example at bottom right.
- A manifest file named `part2_manifest.txt` storing the names of the CSV files produced above, also to be used as input in Part 2.

If a movie record is valid, then you will write the record in its original form to a text file whose name coincides with the movie's genre and ends with the extension `.csv`, as shown in the example at bottom right.

If a movie record is not valid, then you will throw a checked exception related to the error detected and handle the error by writing the following items to the `bad_movie_records.txt` file:

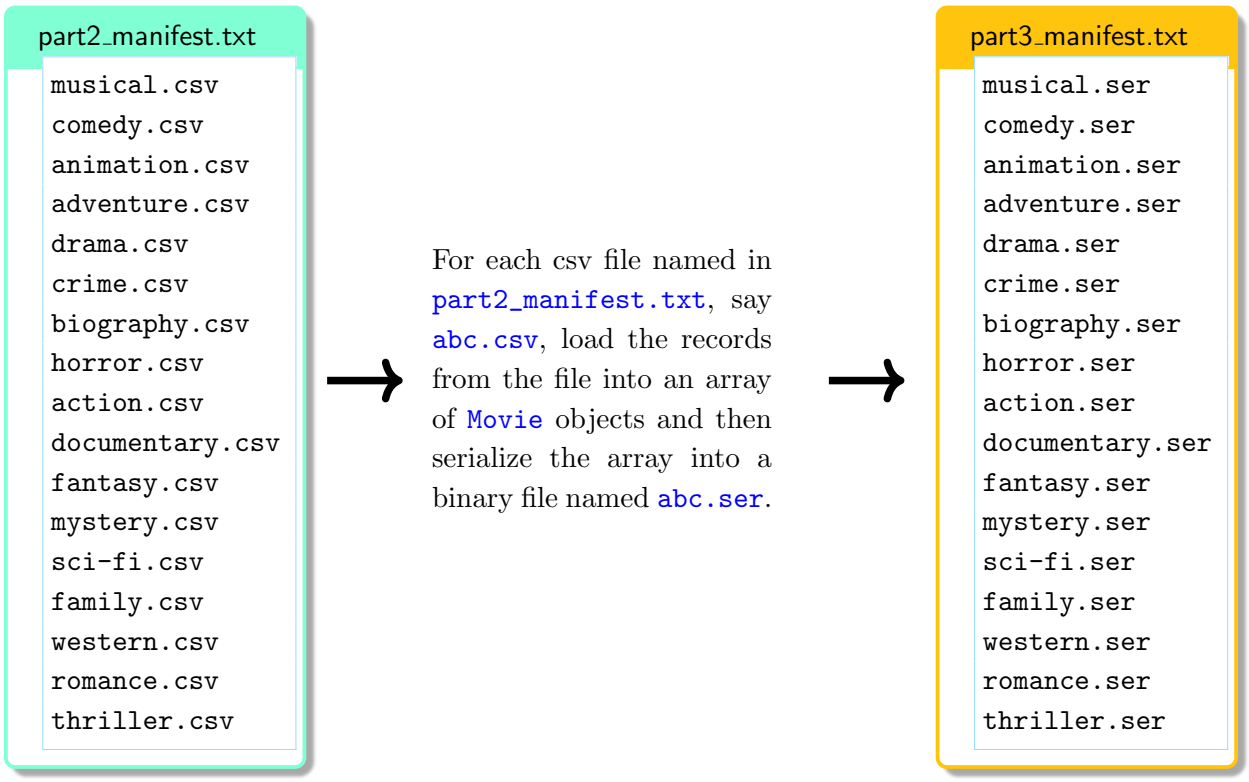
(a) the error and its type, syntax or semantic,
(b) the original movie record,
(c) the name of the input file in which the record appears, and
(d) the record's position (line number) in the input file.

To accomplish that, you will implement the following self-explanatory checked exception classes:[a] `BadYearException`, `BadTitleException`, `BadGenreException`, `BadScoreException`, `BadDurationException`, `BadRatingException`, `BadNameException`, `MissingQuotesException`, `ExcessFieldsException`, `MissingFieldsException`.



[a] See Display 9.4 "A Programmer-Defined Exception Class" in the course textbook.

6 Part 2:
Create and Serialize Arrays of Movie Records



Write a method named `do_part2()` that, for each of the genre files named in `part2_manifest.txt`, will load a `Movie` array with records from the file, and then it will serialize the resulting array into a binary file.[2]

The method should also produce a manifest file named `part3_manifest.txt` that stores the names of the binary files, which are to be used as input in Part 3.

[2]See Display 10.20, File I/O of an Array Object, in the course textbook.

7 Part 3

7.1 Deserialize the Movie Arrays

Write a method named `do_part3()` that will deserialize (reconstruct) the serialized array objects from the binary files named in the file `part3_manifest.txt`.

The deserialization process will produce as many arrays of `Movie` objects as there are binary files named in `part3_manifest.txt`; it will create and return a 2D object of type `Movie[][]` named, say `all_movies`, where `all_movies[0][j]` represents the j'th `Movie` object within the array of all musical `Movie` objects, `all_movies[1][j]` represents the j'th `Movie` object within the array of all comedy `Movie` objects, and so on.

part3_manifest.txt

```
musical.ser
comedy.ser
animation.ser
adventure.ser
drama.ser
crime.ser
biography.ser
horror.ser
action.ser
documentary.ser
fantasy.ser
mystery.ser
sci-fi.ser
family.ser
western.ser
romance.ser
thriller.ser
```

7.2 Navigate the Movie Arrays

Using the 2D movie array from the deserialization process, write a method that will allow the user to interactively navigate through the `Movie` objects referenced by the elements of the movie arrays.

The method will provide a vertical view of the objects within a movie array, allowing the user to display a range of records above or below the *current* movie record at its *current* position, which is initially set to the index of the first record in the array.

Your program must remember the position of the current movie record within each of the movie arrays, allowing the user to switch between the arrays, starting from where they left off previously.

The position of the current movie record within a movie array is adjusted according to whether the range of movie records displayed are above or below it; the adjustment process is detailed on the next page.

Your interactive code must repeatedly display the following menu and perform the selected menu item until the user enters the letter **x** or **X** on the keyboard:

```
----------------------------
          Main Menu
----------------------------
 s  Select a movie array to navigate
 n  Navigate musical movies (0 records)
 x  Exit
----------------------------

Enter Your Choice: s
```

Option n will allow the user to navigate the movie array currently selected, which is initially set to the first movie array.

Option s, after displaying the following sub-menu, will prompt the user to select a movie genre:

```
-----------------------------
        Genre Sub-Menu
-----------------------------
 1  musical                    (0 movies)
 2  comedy                     (73 movies)
 3  animation                  (2 movies)
 4  adventure                  (28 movies)
 5  drama                      (43 movies)
 6  crime                      (11 movies)
 7  biography                  (10 movies)
 8  horror                     (11 movies)
 9  action                     (71 movies)
 10  documentary                (7 movies)
 11  fantasy                    (3 movies)
 12  mystery                    (3 movies)
 13  sci-fi                     (0 movies)
 14  family                     (0 movies)
 15  western                    (0 movies)
 16  romance                    (1 movies)
 17  thriller                   (0 movies)
 18  Exit
-----------------------------
Enter Your Choice: 8
```

The main menu will now display again:

```
-----------------------------
          Main Menu
-----------------------------
 s  Select a movie array to navigate
 n  Navigate horror movies (11 records)
 x  Exit
-----------------------------

Enter Your Choice:
```

Option n opens as follows:

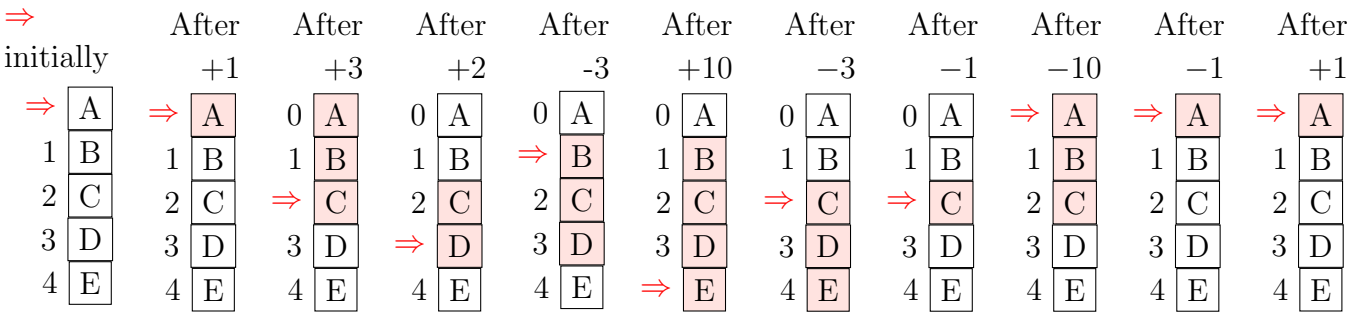
```
Navigating drama  movies (43)
Enter Your Choice:
```

The navigating commands are only integers, say number n.

- If $n = 0$, then the viewing session ends and control will display the main menu again.

- if $n \neq 0$, always display the current movie record, and then
  - If $n < 0$, display $|n| - 1$ movie records above the current position. If there are less than $|n| - 1$ records above the current position, then, after displaying the message `BOF has been reached`, display all the records through the top of the array. In this case, the smallest index within the range of displayed movie records will become the new current movie record.
  - If $n > 0$, display $|n| - 1$ movie records below the current position. If there are less than $|n| - 1$ records below the current position, then, after displaying the message `EOF has been reached`, display all the records through the bottom of the array. In this case, the largest index within the range of displayed movie records will become the new current movie record.

For another example, consider the first array below at left margin where the symbol ⇒ indicates the current movie position and the displayed cells are shown in this color.



8 Requirements

a. You may not use an object of a class in the Java Collections Framework, such as `ArrayList`, `LinkedList`, `HashMap`, `TreeMap`, `HashSet`, `TreeSet`, etc.. Nor may you use any external libraries or existing software to produce what is required.

b. Feel free to introduce your own supporting methods and classes to facilitate your implementation of the operations involved in this assignment.

c. Your program must work for any input files. The CSV files provided with this assignment are only one possible versions, and must not be considered as the general case when writing your code. In addition, the sample results shown on page 10 are based on random movie records, different from your input files.

9 Suggestions

1. Initially, focus on CSV-record processing, not file processing. Start by dissecting a single CSV-record into a string array, hard-coding the CSV-record, or reading it from the keyboard, etc., and constantly varying its contents to challenge your code.

2. Check for syntax errors, handle exceptions, if any

3. Check syntactically valid CSV-records for semantic errors and handle potential exceptions

4. Repeat your CSV-record validation process above, but this time validate the CSV-records from a single movie file, and once the records in that file have been partitioned successfully, repeat the same process with multiple files.

5. Finally, do not be overwhelmed by the number of files involved (3 manifest files, 10 input movie files, and 17 genre files); you prepare only a 10-line manifest file in Part 1, the input movie files are given, and the genre files are generated by your program in Part 1.

10 General Guidelines When Writing Programs

- Include the following comments at the top of your source codes.

```
// -----------------------------------------------------
// Assignment (include number)
// Question: (include question/part number, if applicable)
// Written by: (include your name and student ID)
// -----------------------------------------------------
```

- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.

- Include comments in your program describing the main steps in your program.

- Display a welcome message which includes your name(s).
- Display clear prompts for users when you are expecting the user to enter data from the keyboard.
- All output should be displayed with clear messages and in an easy-to-read format.
- End your program with a closing message so that the user knows that the program has terminated.

11 JavaDoc Documentation

Documentation for your program must be written in `javaDoc`. In addition, the following information must appear at the top of each file:

Name(s) and ID(s)	(include full names and IDs)
COMP249	
Assignment #	(include the assignment number)
Due Date	(include the due date for this assignment)

12 What to submit

Your submission for this assignment must include:

- All Java files related to this assignment,
- all CSV files, both input and output, and
- the error file

13 Evaluation Criteria for Assignment 3 (10 points)

Total	**10 pts**
JavaDoc documentations	1 pt
Producing proper CSV output files in Part 1	2 pt
Producing proper binary files in Part 2	2 pts
Implementing the interactive Part 3	2 pts
Generating and Handling Exceptions	2 pt
General Quality of the Assignment	1 pt