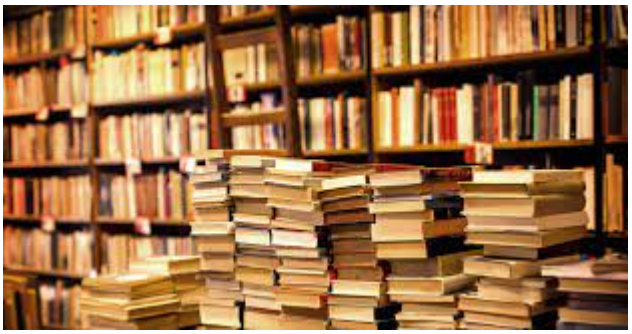
# 1 Purpose

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

# 2 Introduction

A retired employee of a bookstore, Mr. Booker, has a number of old text files he created as a part of his job many years ago.



Each file contains zero or more lines separated by a newline character, with each line storing information about a book, in one of eight book genres related to cartoons, hobbies, movies, music, nostalgia, radio, sports, trains, etc.

With a book catalog that contains almost 2000 entries, Mr. Booker finds it a bit tedious to read through big text files and a bit frustrating when he finds typos in his catalog. Turning to you for help, Mr. Booker would like you to write an interactive program that allows him to navigate through his book titles, categorized by genre, as well as providing a facility for identifying and removing invalid book records. Given that you are now a Java guru, this should be a piece of cake.

# 3 Your Assignment

Write a driver program whose `main()` method implements the requirements of this assignment in three sequentially dependent parts described in the following pages.

```
public static void main(String[] args)
{
   do_part1(); // validating syntax, partition book records based on genre.
   do_part2(); // validating semantics, read the genre files each into arrays of Book objects,
               // then serialize the arrays of Book objects each into binary files.
   do_part3(); // reading the binary files, deserialize the array objects in each file, and
}              // then provide an interacive program to allow the user to navigate the arrays.
```

# 4 CSV-Formatted Input Files

This assignment requires input data from one or more text files, including a file called `Part1_input_file_names.txt` that stores the names of the required input files, one per line, starting at the second line. The first line stores the number, say $n$, of the file names listed below it (16 in the example at right).

The names of the files listed in `Part1_input_file_names.txt` may be any valid file name, and the files themselves 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 display an error message and move on to the next input file, if any.

Each line of an input file represents a book record. Each record contains six data fields and is terminated by a new line character. The data fields themselves are separated by a comma character, a textual format called "CSV" (comma separated values).
The six fields of a book record are:

`title`, `authors`, `price`, `isbn`, `genre`, `year`

For simplicity, you may assume that:

- except for the title field, none of the other data fields may contain commas, and
- the title field itself may not contain double quotes within its text.

In the CSV file format, a field may or may not be enclosed in double quotes. However, when the field itself contains one or more commas, then that field must be enclosed in double quotes.

A book record may contain either syntax or semantic errors:

**Syntax errors** may occur when extracting the data fields of a CSV-formatted input line. An unknown genre is considered a syntax error.

**Semantic errors** may occur when validating the values of the `price`, `isbn`, and `year` fields.

part1_input_file_names.txt

```
16
books1995.csv
books1996.csv
books1997.csv
books1998.csv
books1999.csv
books2000.csv
books2001.csv
books2002.csv
books2003.csv
books2004.csv
books2005.csv
books2006.csv
books2007.csv
books2008.csv
books2009.csv
books2010.csv
```

Syntax Errors

```
Too many fields
Too few fields
Missing field
Unknown genre
```

Semantic Errors

```
invalid ISBN-10
invalid ISBN-13
invalid price
invalid year
```

Here are some examples of CSV-formatted input lines with syntax errors:

```
1 "Manilow, Barry - Biography",Patricia Butler,3.95,0711991979,MRB 2006      Too few fields
2 Manilow, Barry - Biography,Patricia Butler,3.95,0711991979,MRB, 2006       Too many fields
3 "Manilow, Barry - Biography",Patricia Butler,3.95,0711991979,MRB,          Missing field (year)
4 "Manilow, Barry - Biography",Patricia Butler,,0711991979,MRB, 2006         Missing field (price)
```

```
"Manilow, Barry - Biography",Patricia Butler,3.95,0711991979,ABC, 2006
```
unknown genre

Here are some examples of CSV-formatted input lines with semantic errors:

```
"Manilow, Barry - Biography",Patricia Butler,3.95,1711991979,MRB, 2006
```
invalid ISBN-10

```
"Manilow, Barry - Biography",Patricia Butler,3.95,2341711991979,MRB, 2006
```
invalid ISBN-13

```
"Manilow, Barry - Biography",Patricia Butler,3.95,0711991979,MRB, 1006
```
invalid year

```
"Manilow, Barry - Biography",Patricia Butler,-3.95,0711991979,MRB, 2006
```
invalid price

## 4.1 How To Validate Price and Year

Valid prices are non-negative, and valid years fall within the closed range `[1995, 2010]`.

## 4.2 How To Validate an ISBN

- A 10-digit ISBN of the form $x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}$ is valid if the sum $(10x_1 + 9x_2 + 8x_3 + 7x_4 + 6x_5 + 5x_6 + 4x_7 + 3x_8 + 2x_9 + 1x_{10})$ is a multiple of 11.
- A 13-digit ISBN of the form $x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}$ is valid if the sum $(x_1 + 3x_2 + x_3 + 3x_4 + x_5 + 3x_6 + x_7 + 3x_8 + x_9 + 3x_{10} + x_{11} + 3x_{12} + x_{13})$ is a multiple of 10.

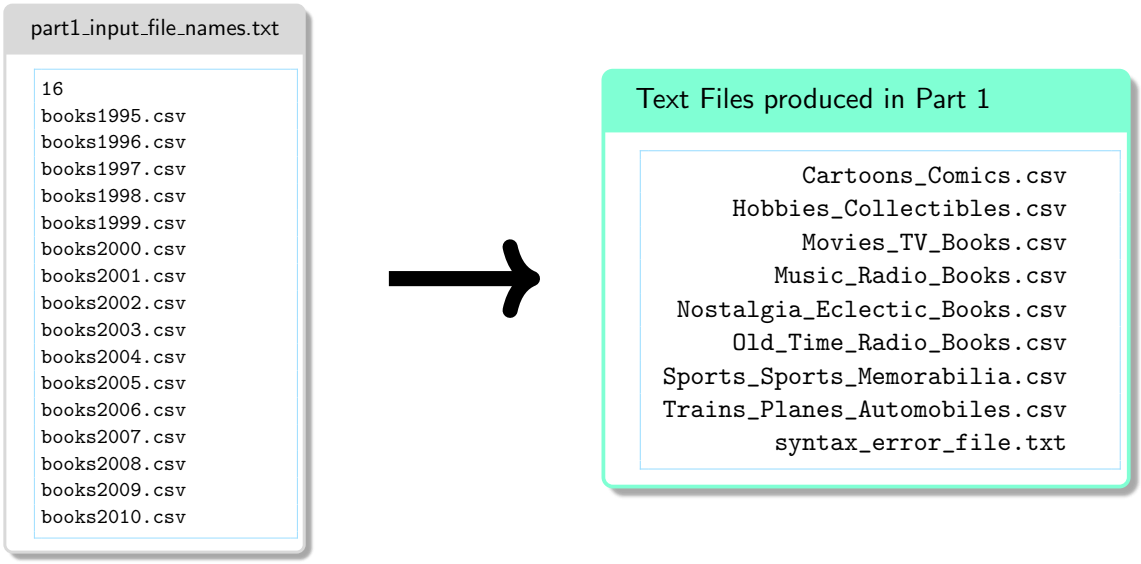
## 4.3 Valid Book Genres, Code, and the Files to Produce

| Genre | Code | Associated File Names |
|---|---|---|
| Cartoons & Comics Books | CCB | `Cartoons_Comics_Books.csv` |
| Hobbies & Collectibles Books | HCB | `Hobbies_Collectibles_Books.csv` |
| Movies & TV | MTV | `Movies_TV.csv` |
| Music & Radio Books | MRB | `Music_Radio_Books.csv` |
| Nostalgia & Eclectic Books | NEB | `Nostalgia_Eclectic_Books.csv` |
| Old-Time Radio | OTR | `Old_Time_Radio.csv` |
| Sports & Sports Memorabilia | SSM | `Sports_Sports_Memorabilia.csv` |
| Trains, Planes & Automobiles | TPA | `Trains_Planes_Automobiles.csv` |
| Bad CSV-formatted Book Recrds | | `syntax_error_file.txt` and `semantic_error_file.txt` |

Note: you may rename .csv files to .csv.txt files if you wish.

# 5 Part 1 of 3

Write a method called `do_part1()` that will read book records from a number of CSV-formatted text files, checking *only* for syntax errors.




Preparing the input files for Part 2, Part 1 will output nine files, eight CSV-formatted text files and a regular text file, whose names are listed in the above box at right.

Since the names and number of the output files and genres may vary depending on the input files, you would ideally create a text file, similar to `Part1_input_file_names.txt` in format, and enter the names and number of the output CSV-files; then, you would use that file as input in Part 2. However, supporting that ideal case would add to the complexity of the program, making it out of the scope of this assignment. Therefore, since your work already involves opening `Part1_input_file_names.txt`, extracting the file names and then processing the corresponding files, there is no need to repeat similar processes in Part 2 and Part 3, so you may hard-code the names of the output files (listed in the green box above) in your program.

Processing the records in each of the CSV-formatted input files (listed in `Part1_input_file_names.txt`), you will encounter records that are either syntactically valid or not.

If a record is syntactically *valid*, then you will write that record to a genre-based CSV-formatted output file (listed in the green box above). You will need to keep track of the size of (the number of book records in) each of the CSV-formatted output files so as to minimize the number of times a single file is opened and closed.

If you encounter a syntactically *invalid* book record, then you will throw an exception corresponding to the syntax error detected, reporting (a) the error, (b) the record, and (c) the name of the file in which the record appears. To accommodate this case, write four checked exception classes[1] called `TooManyFieldsException`, `TooFewFieldsException`, `MissingFieldException`, and `UnknownGenreException`.

Here is an example of the contents of `syntax_error_file.txt` :

[1]Similar to and as simple as Display 9.4 "A Programmer-Defined Exception Class" in the course textbook, as well as the examples/code covered in class.

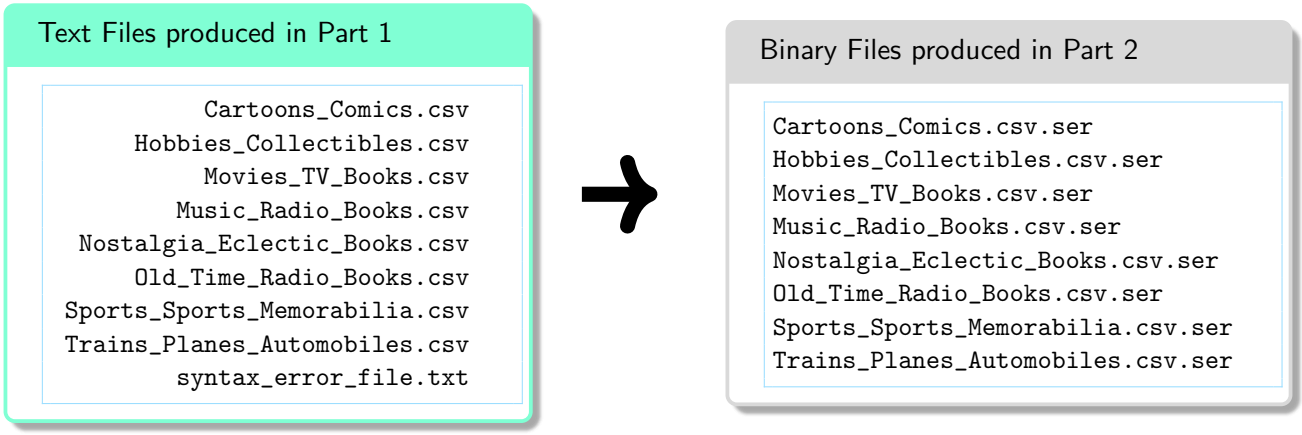
syntax_error_file.txt

```
syntax error in file: books1999.csv
====================
Error: missing title
Record: ,Authors,Price,ISBN,Genre,Released

syntax error in file: books2002.csv
====================
Error: missing price
Record: Celluloid Gaze,Boze Hadleigh,,0879109718,MTV,2002

syntax error in file: books2005.csv
====================
Error: missing authors
Record: Superhero Book,,24.95,1578591546,CCB,2005

syntax error in file: books2009.csv
====================
Error: missing genre
Record: "Rogers, Roy",Robert W. Phillips,35.95,9780786445899,,2009

Error: missing year
Record: Soulful Divas,David Nathan,11.5,0823084302,MRB,

Error:  invalid genre
"Manilow, Barry - Biography",Patricia Butler,3.95,0711991979,ABC, 2006
```

# 6 Part 2 of 3

Recall that except for the file `syntax_error_file.txt`, all the other files listed in the green box below are genre-based CSV-formatted text files containing *syntactically valid* book records.

This part checks each of the syntactically valid book records for semantic errors, setting out to serialize both syntactically and semantically valid book records into binary files whose names appear in the gray box below. (Again, you may hard-code the names of the binary files in your program.)




Write a method called `do_part2()` that reads the genre-based CVS-formatted input text files produced in Part 1, one file at a time, creating an array of valid `Book` objects out of all the semantically valid book records in each input file. A line in an input file, which is already valid syntactically, is processed as follows depending on whether the line contains a semantic error:

(a) The line contains a semantic error rendering it *invalid*:

Throw an exception of the type corresponding to the semantic error detected, reporting the error into an error file named `semantic_error_file.txt` using the same format as that appearing in the file `syntax_error_file.txt`.

To accommodate this case, write four checked exception classes `BadIsbn10Exception`, `BadIsbn13Exception`, `BadPriceException`, `BadYearException`.

(b) The line contains a semantically *valid* book record:

Create a `Book` object based on the data field values of the book record and then store the object in the array of `Book` objects set up for the file containing the valid record.

To accommodate this case, write a simple `Book` class that `implements Serializable` and has six instance variables: `title`, `authors`, `price`, `isbn`, `genre`, and `year`, of which `price` is a `double`, `year` is an `int`, the remaining four instance variables are all of type `String`; initialize the instance variable in a `Book` constructor taking six parameters corresponding to the six instance variables. It should override the `equals()` and `toString()` methods, and for each instance variable, define a pair of corresponding getter and setter methods. Feel free to introduce your own methods to facilitate the implementation of the operations involved in this assignment.

Once you have processed all the records in a file, say `abc.csv`, you will then serialize the resulting array of `Book` objects into a binary file named `abc.csv.ser` and then move on to processing the next input file.

# 7 Part 3 of 3

Write a method called `do_part3()` that will open each of the eight binary files produced in Part 2, deserializing the object in each binary file into an array of `Book` objects. (You may hard-code the names of the binary files in your program if you wish.)

You will then write an interactive code to navigate the objects in any of the arrays of `Book` objects, moving up or down relative to the *current object* in the array, which is initially set to the first object in the array at index 0.

**Binary Files produced in Part 2**

```
Cartoons_Comics.csv.ser
Hobbies_Collectibles.csv.ser
Movies_TV_Books.csv.ser
Music_Radio_Books.csv.ser
Nostalgia_Eclectic_Books.csv.ser
Old_Time_Radio_Books.csv.ser
Sports_Sports_Memorabilia.csv.ser
Trains_Planes_Automobiles.csv.ser
```

The position of the current object is adjusted according to whether the range of objects being viewed (displayed) are above or below the current object; 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
-----------------------------
 v  View the selected file: Cartoons_Comics_Books.csv.ser (4 records)
 s  Select a file to view
 x  Exit
-----------------------------

Enter Your Choice: s
```

Note that "selected file" in the menu effectively refers to the array deserialized from that file.

Option `v` will allow the user to navigate the currently selected file, initially, any one of the arrays.

Option `s` will prompt the user to select a file through the following menu:

```
-----------------------------
        File Sub-Menu
-----------------------------
 1  Cartoons_Comics_Books.csv.ser       (4 records)
 2  Hobbies_Collectibles_Books.csv.ser  (1 records)
 3  Movies_TV.csv.ser                   (6 records)
 4  Music_Radio_Books.csv.ser           (13 records)
 5  Nostalgia_Eclectic_Books.csv.ser    (5 records)
 6  Old_Time_Radio.csv.ser              (0 records)
 7  Sports_Sports_Memorabilia.csv.ser   (0 records)
 8  Trains_Planes_Automobiles.csv.ser   (2 records)
 9  Exit
-----------------------------

Enter Your Choice: 5
```

The main menu will now display again:

```
----------------------------
          Main Menu
----------------------------
 v  View the selected file: Nostalgia_Eclectic_Books.csv.ser (5 records)
 s  Select a file to view
 x  Exit
----------------------------

Enter Your Choice:
```

Option v opens as follows:

```
Enter Your Choice: v
viewing: Nostalgia_Eclectic_Books.csv.ser  (5 records)
```

The viewing commands are only integers, say number $n$, each specifying a range of at most $n$ consecutive records to be displayed, unless $n = 0$.

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

- Whether $n < 0$ or $n > 0$,
  - ⋆ the *current object* in the array is always displayed.
    - ▷ Hence, entering $+1$ or $-1$ will display only the current record.

- If $n > 0$, then the current object and the $(n - 1)$ objects below it, if any, are displayed.
  - ⋆ If there are not $(n - 1)$ records below the current object, then after displaying the last object in the array, the message `EOF has been reached` is displayed.

- The last record in the displayed range will always become the new current object.

  For example, if the current object is at the index, say 17, then for $n = +3$, the current object at index 17 and the $(n-1) = (3-1) = 2$ objects below it at indexes 18, 19, a total of 3 objects, are displayed and the current object will be located at index 19, the last object displayed in the range 17-19.

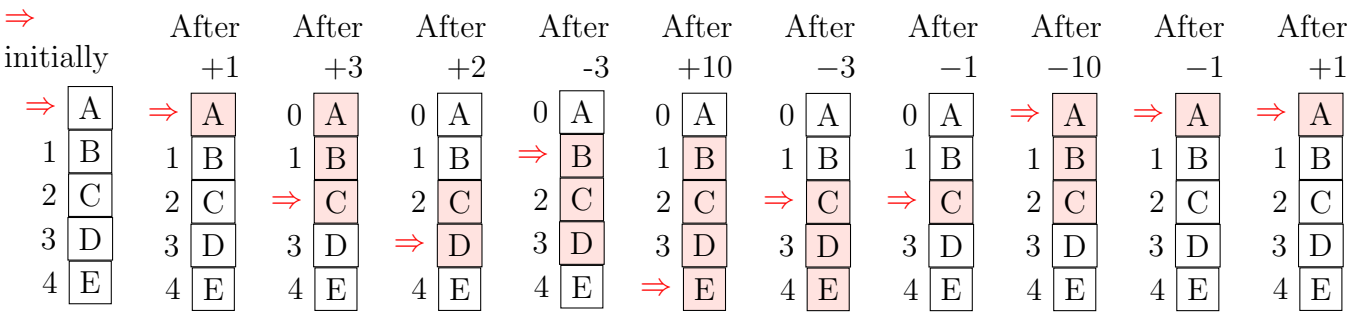
- If $n < 0$, then the current object and the $(|n| - 1)$ objects[2] above it, if any, are displayed.
  - ⋆ If there are not $(|n| - 1)$ objects above the current object, then before displaying the first object in the array, the message `BOF has been reached` is displayed.

  The first object in the displayed range will always become the new current object.

  For example, if the current object is at the index, say 17, then for $n = -3$, the current object at index 17 and the $(|n| - 1) = (|-3| - 1) = 3 - 1 = 2$ objects above it at indexes 15, 16, a total of 3 objects, are displayed and the current object will be located at index 15, the first object displayed in the range 15-17.

[2] $|n|$ denotes the absolute value of $n$

For another example, consider the first array below at left where ⇒ points to the current object and note that the cells displayed in color represent the displayed objects .



# 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. The use of any of these results in an immediate zero mark.

b. For processing of input book records, you may use `String`'s `split` method and/or the `StringTokenizer` class. For array processing, you may use the Java's `Arrays` class.

c. Feel free to introduce and implement classes of your choice to facilitate the implementation of the operations involved in this assignment.

d. You should minimize opening and closing the files.

e. 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. You may rename all `.csv` files to `.csv.txt` files if you wish.

# 9 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.

# 10 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) |

# 11 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 two error files

# 12 Assignment Submission Guidelines

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e., you and one other student).
- Only electronic submissions will be accepted. Zip together the source codes.
- Naming convention for zip file: Create one zip file, containing all source files and produced documentations for your assignment using the following naming convention: The zip file should be called `a#_StudentName_StudentID`, where # is assignment number and `StudentName` and `StudentID` is your name and `ID` number respectively. Use your "official" name only
- no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student `12345678` would submit a zip file named like: `a1_Mike-Simon_123456.zip`. if working in a group, the name should look like: `a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip`.
- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded, and all others will be disregarded.
- If working in a team, only one of the members can upload the assignment. Do NOT upload the file for each of the members!

$\Longrightarrow$ Important: Following your submission, a demo is required (please refer to the course outline for full details). The marker will inform you about the demo times. Please notice that failing to demo your assignment will result in zero mark regardless of your submission. If a demo is

# 13 Now, let us AI the assignment!

Now that you have already finalized your assignment, you are required to plug the entire assignment into an AI tool to generate the entire solution of the assignment for analysis and comparison purposes. You are required to do the following:

a. Go to BassiliChat.com and request the AI tools to generate a solution. **Use Promo Code: PRM8X4ZQ5 to have unlimited access. Please do not share/broadcast this code.**
b. When prompting the tool, ensure that the generated code uses materials covered so far. Do not allow the use of advanced or built-in data structures such as hash tables, ArrayLists, or similar constructs.
c. Obtain solutions from at least two different tools and compare them.
d. In particular, check whether the AI-generated code confirms that your own code worked correctly. If not, identify what the AI produced that is critically different from your code. In all cases, remember that you are still bound by the assignment requirements—even if your code misbehaves. Your task is to compare and comment on the AI results and their differences.
e. In addition to part (d), identify at least two more significant differences between your original solution and the AI-generated one(s). For each difference, decide which version (yours or the AI's) better satisfies the assignment requirements and explain why.
f. Pay special attention to possible "AI hallucinations," which can sometimes be very severe. Also, consider any explanations the tools provide for why the generated code differs from yours, as these may justify the code they produced.
g. Document all the prompts and search commands you used. Clearly highlight any misleading prompts you may have given, and include a summary of the misbehaviors (if any), along with at least two major differences identified in part (e). **Your AI Appendix should NOT exceed 20 pages including 2 page final summary; so make sure that it is precise and to the point.**

# 14 Evaluation Criteria for Assignment 2 (10 points)

| **Total** | **10 pts** |
|---|---|
| JavaDoc documentations | 0.5 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 |
| AI Report | 1.5 pt |