Documentation Practical Exercise 2

Student:

- Manuel Buser

Degree:

- MSc Computer Science

Module:

- Multimedia Retrieval

1 First step Import the CSV and Create an Index

```java
public class Indexer {
    1 usage
    private static final String INDEX_DIR = "src/main/resources/lucene-index";

    Edit | Explain | Test | Document | Fix
    1 usage  new *
    public void createIndex(String csvFilePath) {
        try {
            // Clear the index directory to avoid conflicts
            Path path  = Paths.get(INDEX_DIR);
            if (Files.exists(path)) {
                for (var file : Objects.requireNonNull(path.toFile().listFiles())) {
                    if (file.isFile()) file.delete();
                }
            }

            FSDirectory directory = FSDirectory.open(path);
            StandardAnalyzer analyzer = new StandardAnalyzer();
            IndexWriterConfig config = new IndexWriterConfig(analyzer);
            IndexWriter writer = new IndexWriter(directory, config);

            BufferedReader br = new BufferedReader(new FileReader(csvFilePath));
            String line = br.readLine(); // Skip header

            while ((line = br.readLine()) != null) {
                String[] fields = line.split( regex: ",(?=([^\"]*\"[^\"]*\")*[^\"]*$)");
                if (fields.length > 7) { // Ensure sufficient fields exist
                    Document doc = new Document();
                    doc.add(new TextField( name: "title", fields[1].trim(), Field.Store.YES));
                    doc.add(new TextField( name: "year", fields[2].trim(), Field.Store.YES));
                    doc.add(new TextField( name: "genre", fields[5].trim(), Field.Store.YES));
                    doc.add(new TextField( name: "overview", fields[7].trim(), Field.Store.YES));
                    writer.addDocument(doc);

                    System.out.println("Indexed: " + fields[1].trim());
                }
            }

            writer.close();
            System.out.println("Indexing completed successfully!");

        } catch (Exception e) {
```

First, I imported the CSV that I downloaded as a ZIP File and Extracted it to the resource folder inside my Maven Project. Then we created the index.

2 Building a basic search function

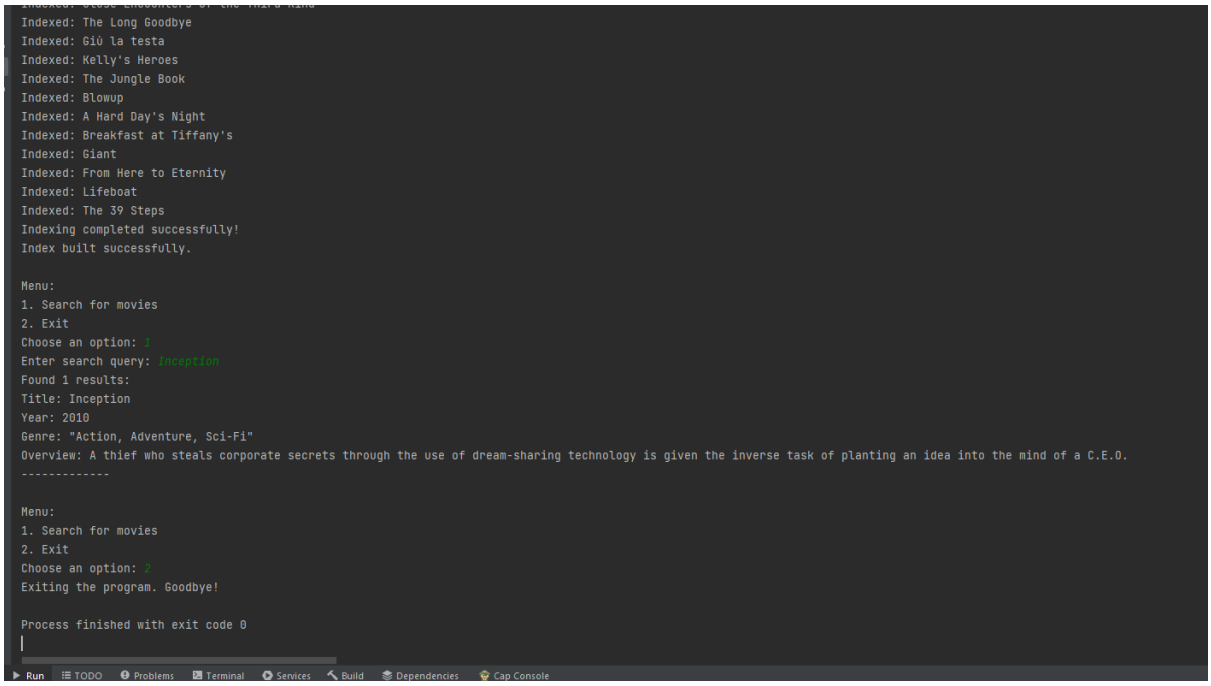
```java
2 usages  new *
public class Searcher {
    1 usage
    private static final String INDEX_DIR = "src/main/resources/lucene-index";

    Edit | Explain | Test | Document | Fix
    1 usage  new *
    public void search(String queryStr) {
        try {
            FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
            DirectoryReader reader = DirectoryReader.open(directory);
            IndexSearcher searcher = new IndexSearcher(reader);
            StandardAnalyzer analyzer = new StandardAnalyzer();

            QueryParser parser = new QueryParser( f: "title", analyzer);
            parser.setAllowLeadingWildcard(true); // Enable partial match
            Query query = parser.parse( query: "*" + queryStr + "*");

            TopDocs results = searcher.search(query,  n: 10);
            ScoreDoc[] hits = results.scoreDocs;

            System.out.println("Found " + hits.length + " results:");
            for (ScoreDoc hit : hits) {
                Document doc = searcher.doc(hit.doc);
                System.out.println("Title: " + doc.get("title"));
                System.out.println("Year: " + doc.get("year"));
                System.out.println("Genre: " + doc.get("genre"));
                System.out.println("Overview: " + doc.get("overview"));
                System.out.println("-------------");
            }

            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

In this basic search function, we can search for movie titles, and it will find it if it found a direct match.

To run this, I built a basic main class, with a simple user text-based user interface in the console:

```java
import java.util.Scanner;

Edit | Explain | Test | Document | Fix
no usages  new *
public class Main {
    Edit | Explain | Test | Document | Fix
    no usages  new *
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Indexer indexer = new Indexer();
        Searcher searcher = new Searcher();

        System.out.println("Welcome to IMDB Movie Search!");
        System.out.println("Building index...");
        indexer.createIndex( csvFilePath: "src/resources/imdb_top_1000.csv");
        System.out.println("Index built successfully.");

        boolean running = true;

        while (running) {
            System.out.println("\nMenu:");
            System.out.println("1. Search for movies");
            System.out.println("2. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline

            switch (choice) {
                case 1:
                    System.out.print("Enter search query: ");
                    String query = scanner.nextLine();
                    searcher.search(query);
                    break;
                case 2:
                    System.out.println("Exiting the program. Goodbye!");
                    running = false;
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }

        scanner.close();
    }
```

To test this first functionality, I ran the program:



Next step was to extend the search for direct year and genre match. The method is enhanced to allow the user to specify which field to search in and provides the same robust partial matching functionality.

```
Edit | Explain | Test | Document | Fix
2 usages  manuu1999
public class Searcher {
    1 usage
    private static final String INDEX_DIR = "src/main/resources/lucene-index";

    Edit | Explain | Test | Document | Fix
    1 usage  manuu1999
    public void search(String field, String queryStr) {
        try {
            FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
            DirectoryReader reader = DirectoryReader.open(directory);
            IndexSearcher searcher = new IndexSearcher(reader);
            StandardAnalyzer analyzer = new StandardAnalyzer();

            QueryParser parser = new QueryParser(field, analyzer);
            parser.setAllowLeadingWildcard(true); // Enable partial match
            Query query = parser.parse( query: "*" + queryStr + "*");

            TopDocs results = searcher.search(query,  n: 10);
            ScoreDoc[] hits = results.scoreDocs;

            System.out.println("Found " + hits.length + " results:");
            for (ScoreDoc hit : hits) {
                Document doc = searcher.doc(hit.doc);
                System.out.println("Title: " + doc.get("title"));
                System.out.println("Year: " + doc.get("year"));
                System.out.println("Genre: " + doc.get("genre"));
                System.out.println("Overview: " + doc.get("overview"));
                System.out.println("-------------");
            }

            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
boolean running = true;

while (running) {
    System.out.println("\nMenu:");
    System.out.println("1. Search for movies");
    System.out.println("2. Exit");
    System.out.print("Choose an option: ");

    int choice = scanner.nextInt();
    scanner.nextLine(); // Consume the newline

    switch (choice) {
        case 1:
            System.out.println("\nSearch by:");
            System.out.println("1. Title");
            System.out.println("2. Year");
            System.out.println("3. Genre");
            System.out.print("Choose a search field: ");
            int fieldChoice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline

            String field;
            switch (fieldChoice) {
                case 1:
                    field = "title";
                    break;
                case 2:
                    field = "year";
                    break;
                case 3:
                    field = "genre";
                    break;
                default:
                    System.out.println("Invalid field choice. Returning to menu.");
                    continue;
            }

            System.out.print("Enter search query: ");
            String query = scanner.nextLine();
            searcher.search(field, query);
            break;

        case 2:
            System.out.println("Exiting the program. Goodbye!");
            running = false;
            break;

        default:
            System.out.println("Invalid option. Please try again.");
    }
}
```

Output:


Indexing completed successfully!
Index built successfully.
Menu:
1. Search for movies
2. Exit
Choose an option: 1
Search by:
1. Title
2. Year
3. Genre
Choose a search field: 2
Enter search query: 2010
Found 10 results:
Title: Inception
Year: 2010
Genre: "Action, Adventure, Sci-Fi"
Overview: A thief who steals corporate secrets through the use of dream-sharing technology is given the inverse task of planting an idea into the mind of a C.E.O.
Title: Incendies
Year: 2010
Genre: "Drama, Mystery, War"
Overview: Twins journey to the Middle East to discover their family history and fulfill their mother's last wishes.
Title: Udaan
Year: 2010
Genre: Drama
Overview: "Expelled from his school, a 16-year old boy returns home to his abusive and oppressive father."
Title: Shutter Island
Year: 2010
Genre: "Mystery, Thriller"
Overview: "In 1954, a U.S. Marshal investigates the disappearance of a murderer who escaped from a hospital for the criminally insane."
Title: Toy Story 3
Year: 2010
Genre: "Animation, Adventure, Comedy"
Overview: "The toys are mistakenly delivered to a day-care center instead of the attic right before Andy leaves for college, and it's up to Woody to convince the other toys that they weren't
Title: How to Train Your Dragon
Year: 2010

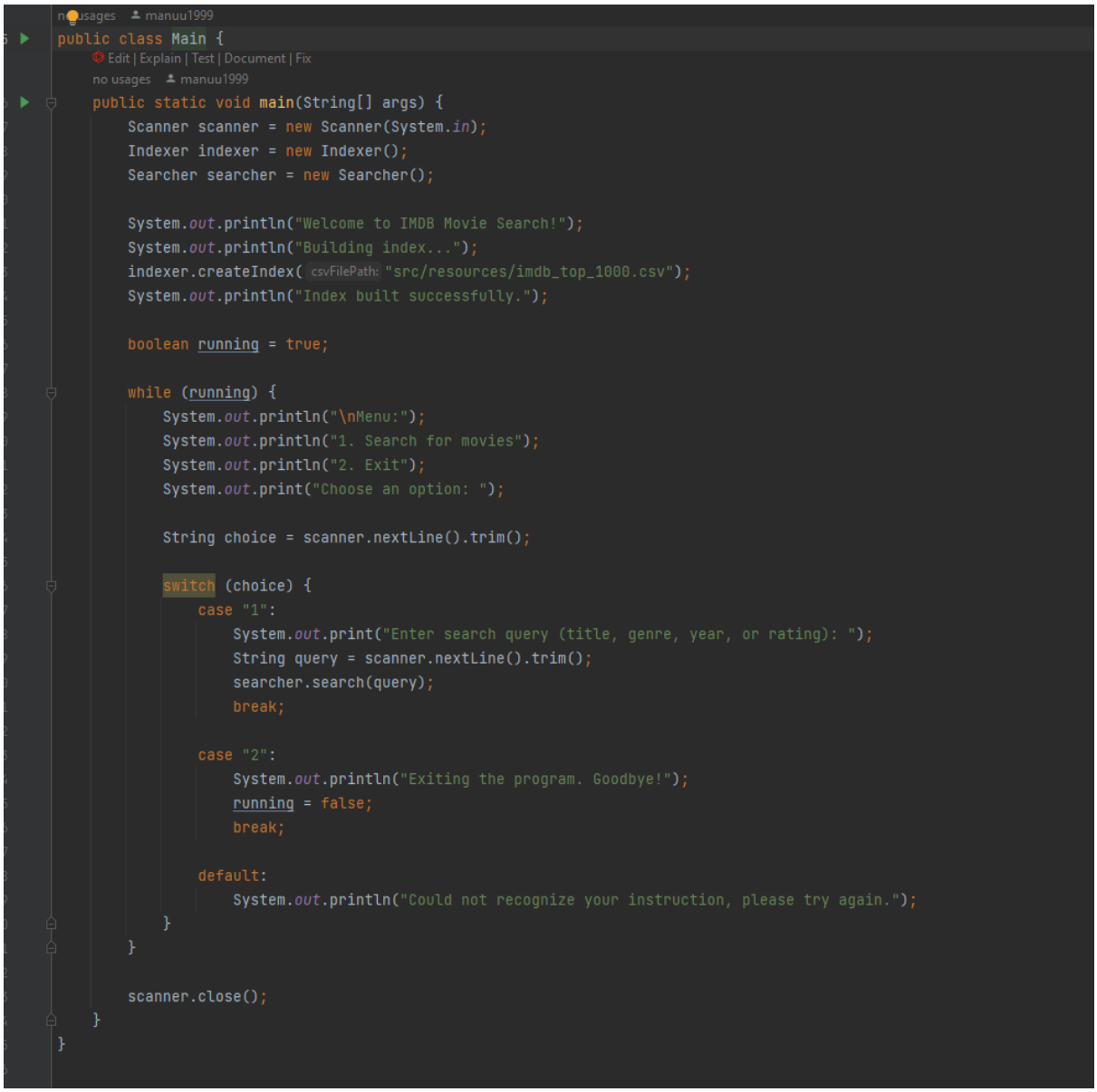
3 Enhance the search results by considering more relevance factors

Here we combine all search categories into one loop and the program should recognize itself for what to look for based on the input:

```
1 usage  manuu1999 *
public void search(String queryStr) {
    try {
        FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
        DirectoryReader reader = DirectoryReader.open(directory);
        IndexSearcher searcher = new IndexSearcher(reader);
        StandardAnalyzer analyzer = new StandardAnalyzer();

        // Define field boosts
        Map<String, Float> boosts = new HashMap<>();
        boosts.put("title", 2.0f);  // Higher relevance for title
        boosts.put("genre", 1.0f);
        boosts.put("year", 1.0f);
        boosts.put("rating", 1.0f);

        // Create MultiFieldQueryParser with field boosts
        MultiFieldQueryParser parser = new MultiFieldQueryParser(
                boosts.keySet().toArray(new String[0]), analyzer
        );
        parser.setAllowLeadingWildcard(true); // Enable partial match

        Query query = parser.parse(queryStr);

        TopDocs results = searcher.search(query,  n: 10);
        ScoreDoc[] hits = results.scoreDocs;

        System.out.println("Found " + hits.length + " results:");
        for (ScoreDoc hit : hits) {
            Document doc = searcher.doc(hit.doc);
            System.out.println("Title: " + doc.get("title"));
            System.out.println("Year: " + doc.get("year"));
            System.out.println("Genre: " + doc.get("genre"));
            System.out.println("Rating: " + doc.get("rating"));
            System.out.println("Overview: " + doc.get("overview"));
            System.out.println("-------------");
        }
```

The boosting mechanism implemented in the above code addresses improving the ranking of search results by considering the occurrences of keywords in different fields with varying levels of importance.

Here – reduce the loop:



```
n usages  manuu1999
public class Main {
    Edit | Explain | Test | Document | Fix
    no usages  manuu1999
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Indexer indexer = new Indexer();
        Searcher searcher = new Searcher();

        System.out.println("Welcome to IMDB Movie Search!");
        System.out.println("Building index...");
        indexer.createIndex( csvFilePath: "src/resources/imdb_top_1000.csv");
        System.out.println("Index built successfully.");

        boolean running = true;

        while (running) {
            System.out.println("\nMenu:");
            System.out.println("1. Search for movies");
            System.out.println("2. Exit");
            System.out.print("Choose an option: ");

            String choice = scanner.nextLine().trim();

            switch (choice) {
                case "1":
                    System.out.print("Enter search query (title, genre, year, or rating): ");
                    String query = scanner.nextLine().trim();
                    searcher.search(query);
                    break;

                case "2":
                    System.out.println("Exiting the program. Goodbye!");
                    running = false;
                    break;

                default:
                    System.out.println("Could not recognize your instruction, please try again.");
            }
        }

        scanner.close();
    }
}
```

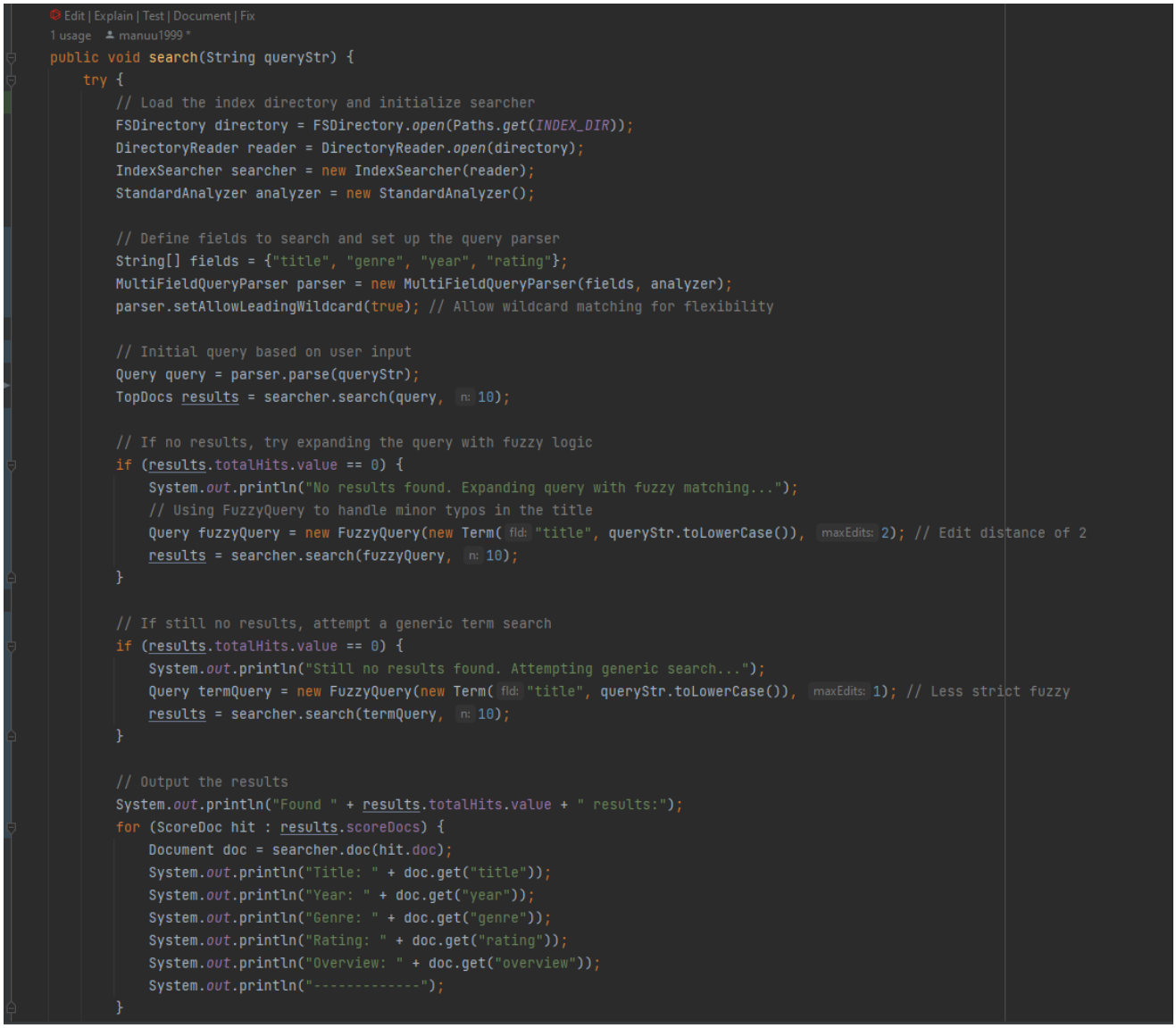
Output:

```
Indexed: Lifeboat
Indexed: The 39 Steps
Indexing completed successfully!
Index built successfully.

Menu:
1. Search for movies
2. Exit
Choose an option: 1
Enter search query (title, genre, year, or rating): inception
Found 1 results:
Title: Inception
Year: 2010
Genre: "Action, Adventure, Sci-Fi"
Rating: 8.8
Overview: A thief who steals corporate secrets through the use of dream-sharing technology is given the inverse task of planting an
-------------

Menu:
1. Search for movies
2. Exit
Choose an option: 1
Enter search query (title, genre, year, or rating): 2010
Found 10 results:
Title: Inception
Year: 2010
Genre: "Action, Adventure, Sci-Fi"
```

```
Rating: 8
Overview: "A committed dancer struggles to maintain her sanity after winning the lead role in a production of Tchaikovsky's ""Swan Lake""."
-------------

Menu:
1. Search for movies
2. Exit
Choose an option: 1
Enter search query (title, genre, year, or rating): 8.5
Found 10 results:
Title: Joker
Year: 2019
Genre: "Crime, Drama, Thriller"
Rating: 8.5
Overview: "In Gotham City, mentally troubled comedian Arthur Fleck is disregarded and mistreated by society. He then embarks on a downward spiral of revolution and bloody crime. Th
-------------
Title: Whiplash
```

```
-------------

Menu:
1. Search for movies
2. Exit
Choose an option: 1
Enter search query (title, genre, year, or rating): comedy
Found 10 results:
Title: The King of Comedy
Year: 1982
Genre: "Comedy, Crime, Drama"
Rating: 7.8
Overview: "Rupert Pupkin is a passionate yet unsuccessful comic who craves nothing more than to be in the spotlight and to achieve this, he stalks and
-------------
Title: Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb
Year: 1964
Genre: Comedy
Rating: 8.4
Overview: An insane general triggers a path to nuclear holocaust that a War Room full of politicians and generals frantically tries to stop.
-------------
Title: Life of Brian
Year: 1979
Genre: Comedy
Rating: 8.1
```

4 Enhance query term matching with query expansion



```
Edit | Explain | Test | Document | Fix
1 usage  manuu1999 *
public void search(String queryStr) {
    try {
        // Load the index directory and initialize searcher
        FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
        DirectoryReader reader = DirectoryReader.open(directory);
        IndexSearcher searcher = new IndexSearcher(reader);
        StandardAnalyzer analyzer = new StandardAnalyzer();

        // Define fields to search and set up the query parser
        String[] fields = {"title", "genre", "year", "rating"};
        MultiFieldQueryParser parser = new MultiFieldQueryParser(fields, analyzer);
        parser.setAllowLeadingWildcard(true); // Allow wildcard matching for flexibility

        // Initial query based on user input
        Query query = parser.parse(queryStr);
        TopDocs results = searcher.search(query,  n: 10);

        // If no results, try expanding the query with fuzzy logic
        if (results.totalHits.value == 0) {
            System.out.println("No results found. Expanding query with fuzzy matching...");
            // Using FuzzyQuery to handle minor typos in the title
            Query fuzzyQuery = new FuzzyQuery(new Term( fld: "title", queryStr.toLowerCase()),  maxEdits: 2); // Edit distance of 2
            results = searcher.search(fuzzyQuery,  n: 10);
        }

        // If still no results, attempt a generic term search
        if (results.totalHits.value == 0) {
            System.out.println("Still no results found. Attempting generic search...");
            Query termQuery = new FuzzyQuery(new Term( fld: "title", queryStr.toLowerCase()),  maxEdits: 1); // Less strict fuzzy
            results = searcher.search(termQuery,  n: 10);
        }

        // Output the results
        System.out.println("Found " + results.totalHits.value + " results:");
        for (ScoreDoc hit : results.scoreDocs) {
            Document doc = searcher.doc(hit.doc);
            System.out.println("Title: " + doc.get("title"));
            System.out.println("Year: " + doc.get("year"));
            System.out.println("Genre: " + doc.get("genre"));
            System.out.println("Rating: " + doc.get("rating"));
            System.out.println("Overview: " + doc.get("overview"));
            System.out.println("-------------");
        }
```

Here I made a progressive search strategy, starting with the exact query and expanding to fuzzy matching if no results are found.

Output: Looking for Inception but tipping in “Inciption”:

5 Implement faceted search to allow users to filter result

```java
public class Searcher {
    1 usage
    private static final String INDEX_DIR = "src/main/resources/lucene-index";

    Edit | Explain | Test | Document | Fix
    // Perform the initial search
    1 usage  manuu1999 *
    public List<Document> search(String queryStr) {
        List<Document> resultsList = new ArrayList<>();
        try {
            FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
            DirectoryReader reader = DirectoryReader.open(directory);
            IndexSearcher searcher = new IndexSearcher(reader);
            StandardAnalyzer analyzer = new StandardAnalyzer();

            // Fields to search
            String[] fields = {"title", "genre", "year", "rating"};
            MultiFieldQueryParser parser = new MultiFieldQueryParser(fields, analyzer);
            parser.setAllowLeadingWildcard(true);

            Query query = parser.parse(queryStr);
            TopDocs results = searcher.search(query,  n: 100);

            System.out.println("Found " + results.totalHits.value + " results:");
            for (ScoreDoc hit : results.scoreDocs) {
                Document doc = searcher.doc(hit.doc);
                resultsList.add(doc);

                System.out.println("Title: " + doc.get("title"));
                System.out.println("Year: " + doc.get("year"));
                System.out.println("Genre: " + doc.get("genre"));
                System.out.println("Rating: " + doc.get("rating"));
                System.out.println("Overview: " + doc.get("overview"));
                System.out.println("-------------");
            }

            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return resultsList;
    }
```

Next to the regular search function I implemented an additional function (Next Page), that can narrow down the result with additional filters. So, the new workflow will be like: First search in the same way as before. Then these results are also **stored in a List<Document>** called currentResults for further processing. If the user chooses to filter, they input an additional term (e.g., comedy). Then the program takes the currentResults and narrows it down with the new function.

```
Edit | Explain | Test | Document | Fix
// Narrow down the results based on a filter
1 usage  manuu1999 *
public List<Document> filterResults(List<Document> currentResults, String filter) {
    List<Document> filteredResults = currentResults.stream()
            .filter(doc -> doc.get("title").toLowerCase().contains(filter.toLowerCase())
                    || doc.get("genre").toLowerCase().contains(filter.toLowerCase())
                    || doc.get("year").equals(filter)
                    || doc.get("rating").equals(filter))
            .collect(Collectors.toList());

    if (filteredResults.isEmpty()) {
        System.out.println("No results found after filtering.");
    } else {
        System.out.println("Filtered results:");
        for (Document doc : filteredResults) {
            System.out.println("Title: " + doc.get("title"));
            System.out.println("Year: " + doc.get("year"));
            System.out.println("Genre: " + doc.get("genre"));
            System.out.println("Rating: " + doc.get("rating"));
            System.out.println("Overview: " + doc.get("overview"));
            System.out.println("-------------");
        }
    }

    return filteredResults;
}
}
```

Output:

First, we make a normal search that works exactly as in the last step.

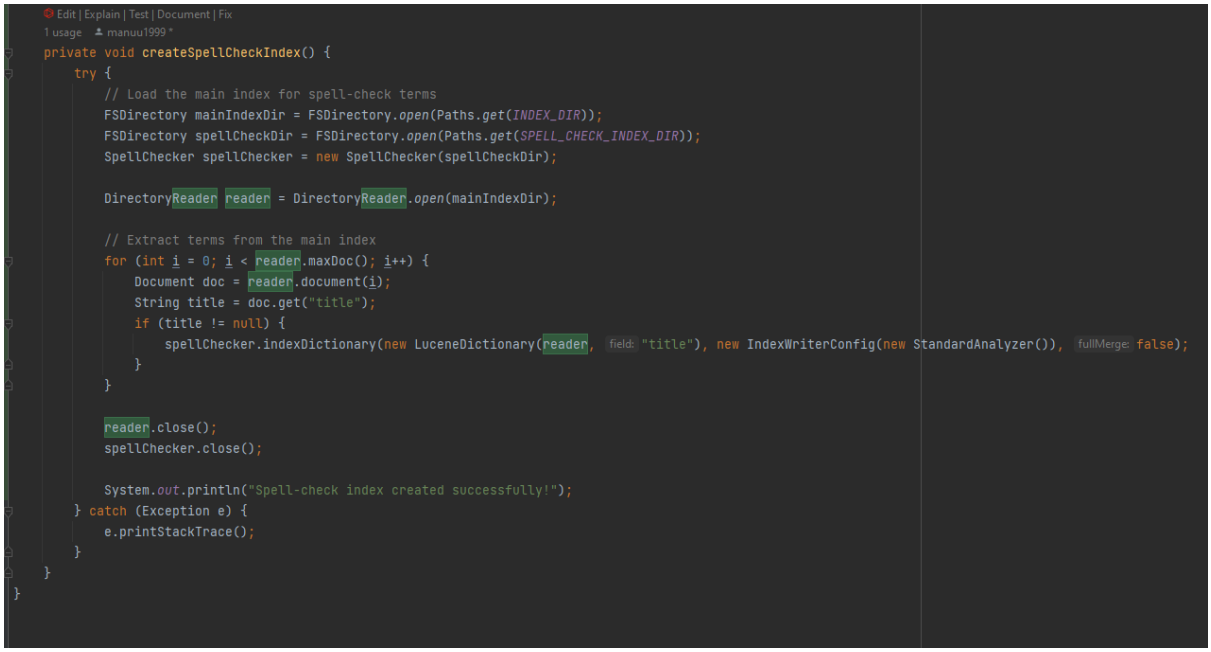
But then after the first search another option appears in the menu:

We can input another search term. Above we searched first for movies with a rating over 8.5 and now we narrow down the result with an additional filter set to comedy. After that we will get the same list as before but filtered with the additional filter:

As it is visible above, you can run this filter as many times as you want, and the Output List gets smaller and smaller. Here we inputted additionally the year 1931, so after one normal search and two filters we found the movie with the Title City Lights that matches the 8.5 Rating, the Comedy Genre and the release year 1931.

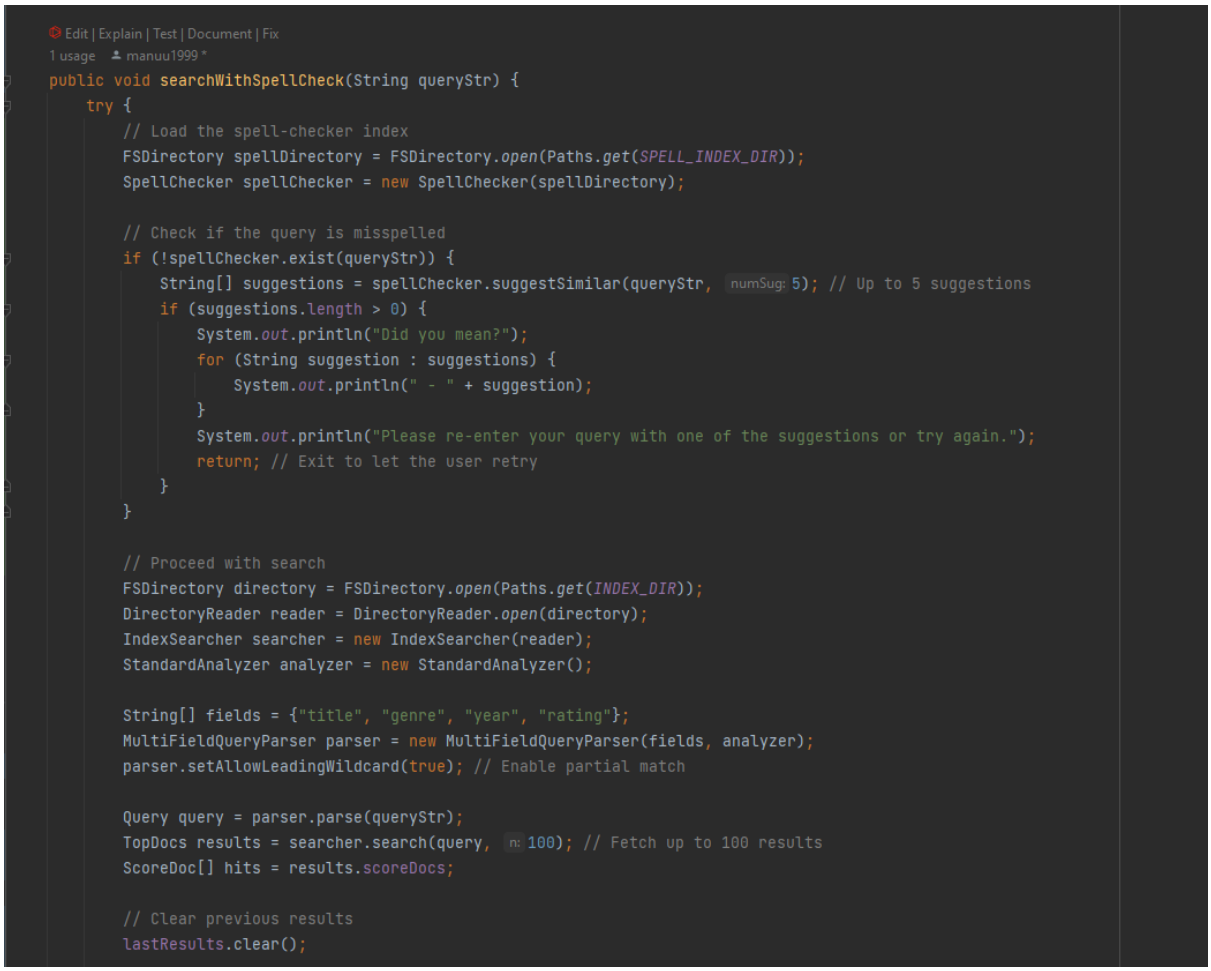
6 Add spell-checking

I created a new spellcheck Index:



```
Edit | Explain | Test | Document | Fix
1 usage  manuu1999 *
private void createSpellCheckIndex() {
    try {
        // Load the main index for spell-check terms
        FSDirectory mainIndexDir = FSDirectory.open(Paths.get(INDEX_DIR));
        FSDirectory spellCheckDir = FSDirectory.open(Paths.get(SPELL_CHECK_INDEX_DIR));
        SpellChecker spellChecker = new SpellChecker(spellCheckDir);

        DirectoryReader reader = DirectoryReader.open(mainIndexDir);

        // Extract terms from the main index
        for (int i = 0; i < reader.maxDoc(); i++) {
            Document doc = reader.document(i);
            String title = doc.get("title");
            if (title != null) {
                spellChecker.indexDictionary(new LuceneDictionary(reader,  field: "title"), new IndexWriterConfig(new StandardAnalyzer()),  fullMerge: false);
            }
        }

        reader.close();
        spellChecker.close();

        System.out.println("Spell-check index created successfully!");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}
```

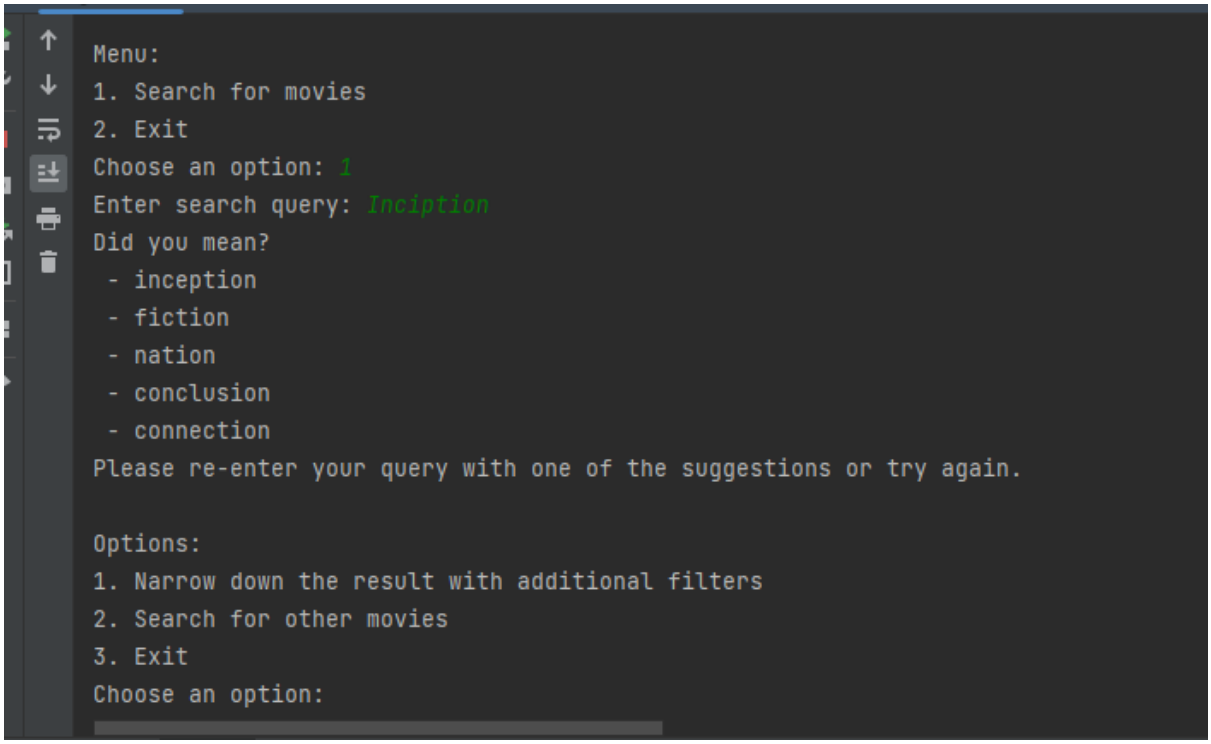
And then implemented this in the search function:



```
Edit | Explain | Test | Document | Fix
1 usage  manuu1999 *
public void searchWithSpellCheck(String queryStr) {
    try {
        // Load the spell-checker index
        FSDirectory spellDirectory = FSDirectory.open(Paths.get(SPELL_INDEX_DIR));
        SpellChecker spellChecker = new SpellChecker(spellDirectory);

        // Check if the query is misspelled
        if (!spellChecker.exist(queryStr)) {
            String[] suggestions = spellChecker.suggestSimilar(queryStr,  numSug: 5); // Up to 5 suggestions
            if (suggestions.length > 0) {
                System.out.println("Did you mean?");
                for (String suggestion : suggestions) {
                    System.out.println(" - " + suggestion);
                }
                System.out.println("Please re-enter your query with one of the suggestions or try again.");
                return; // Exit to let the user retry
            }
        }

        // Proceed with search
        FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
        DirectoryReader reader = DirectoryReader.open(directory);
        IndexSearcher searcher = new IndexSearcher(reader);
        StandardAnalyzer analyzer = new StandardAnalyzer();

        String[] fields = {"title", "genre", "year", "rating"};
        MultiFieldQueryParser parser = new MultiFieldQueryParser(fields, analyzer);
        parser.setAllowLeadingWildcard(true); // Enable partial match

        Query query = parser.parse(queryStr);
        TopDocs results = searcher.search(query,  n: 100); // Fetch up to 100 results
        ScoreDoc[] hits = results.scoreDocs;

        // Clear previous results
        lastResults.clear();
```

Output:

Here we see the implementation of the spell checking in action:



7 Implement pagination for search results

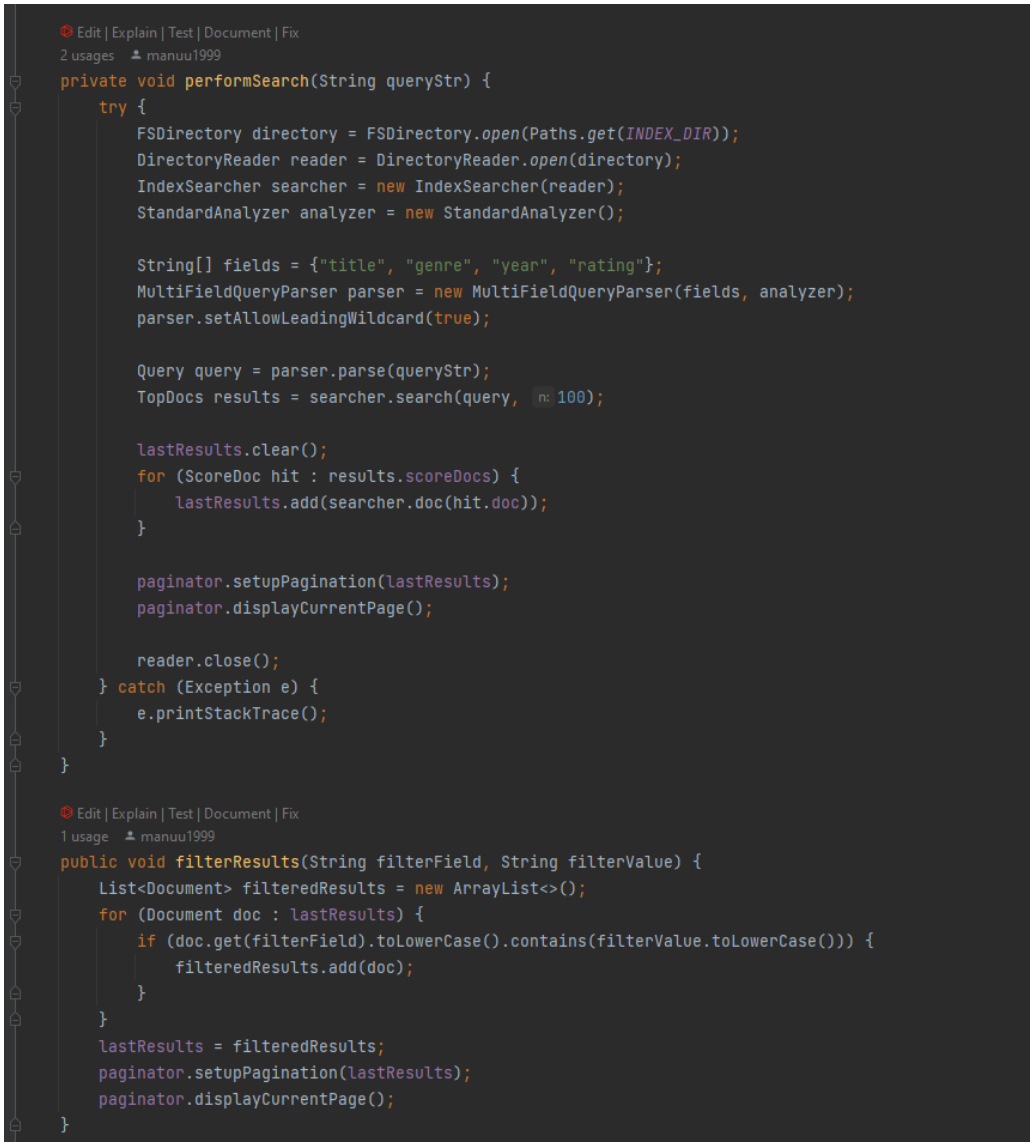
For this last task I had to make a new class with some additional help functions and variables:

```
import java.util.List;

Edit | Explain | Test | Document | Fix
2 usages
public class Paginator {
    7 usages
    private int currentPage = 1;
    5 usages
    private static final int RESULTS_PER_PAGE = 10; // Show 10 results per page
    8 usages
    private List<Document> results = new ArrayList<>(); // Ensure it's initialized

    Edit | Explain | Test | Document | Fix
    2 usages
    public void setupPagination(List<Document> results) {
        this.results = results;
        this.currentPage = 1;
    }

    Edit | Explain | Test | Document | Fix
    4 usages
    public void displayCurrentPage() {
        if (results == null || results.isEmpty()) {
            System.out.println("No results to display.");
            return;
        }

        int start = (currentPage - 1) * RESULTS_PER_PAGE;
        int end = Math.min(start + RESULTS_PER_PAGE, results.size());

        System.out.println("\nPage " + currentPage + " of " + ((results.size() + RESULTS_PER_PAGE - 1) / RESULTS_PER_PAGE));
        for (int i = start; i < end; i++) {
            Document doc = results.get(i);
            System.out.println("Title: " + doc.get("title"));
            System.out.println("Year: " + doc.get("year"));
            System.out.println("Genre: " + doc.get("genre"));
            System.out.println("Rating: " + doc.get("rating"));
            System.out.println("Overview: " + doc.get("overview"));
            System.out.println("-------------");
        }

        if (end < results.size()) {
            System.out.println("...more available on the next page.");
        }
    }
```

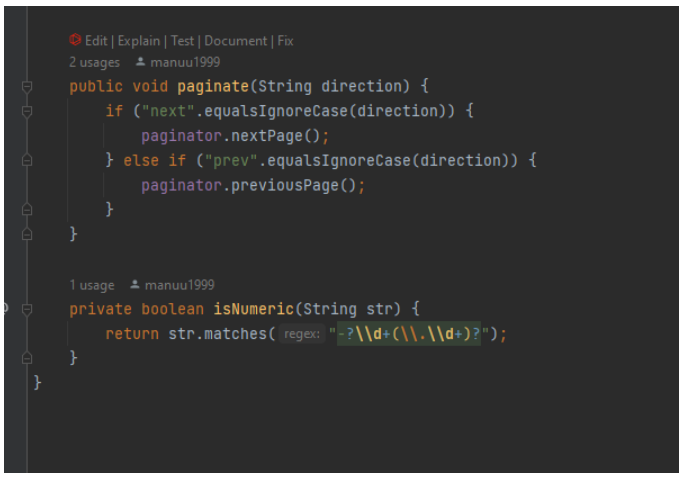
```
    Edit | Explain | Test | Document | Fix
    1 usage
    public void nextPage() {
        if ((currentPage * RESULTS_PER_PAGE) < results.size()) {
            currentPage++;
            displayCurrentPage();
        } else {
            System.out.println("You are already on the last page.");
        }
    }

    Edit | Explain | Test | Document | Fix
    1 usage
    public void previousPage() {
        if (currentPage > 1) {
            currentPage--;
            displayCurrentPage();
        } else {
            System.out.println("You are already on the first page.");
        }
    }
}
```

Here in the searcher class, I adapted to the new functionality:

```
2 usages   manuu1999
public class Searcher {
    1 usage
    private static final String INDEX_DIR = "src/main/resources/lucene-index";
    1 usage
    private static final String SPELL_INDEX_DIR = "src/main/resources/spellcheck-index";

    6 usages
    private List<Document> lastResults = new ArrayList<>();
    6 usages
    private Paginator paginator = new Paginator();

    Edit | Explain | Test | Document | Fix
    1 usage   manuu1999
    public void searchWithSpellCheck(String queryStr) {
        try {
            if (isNumeric(queryStr)) {
                // Skip spell checking for numeric inputs
                performSearch(queryStr);
                return;
            }

            FSDirectory spellDirectory = FSDirectory.open(Paths.get(SPELL_INDEX_DIR));
            SpellChecker spellChecker = new SpellChecker(spellDirectory);

            if (!spellChecker.exist(queryStr)) {
                String[] suggestions = spellChecker.suggestSimilar(queryStr, numSug: 5);
                if (suggestions.length > 0) {
                    System.out.println("Did you mean?");
                    for (String suggestion : suggestions) {
                        System.out.println(" - " + suggestion);
                    }
                    return;
                }
            }

            performSearch(queryStr);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```



```
Edit | Explain | Test | Document | Fix
2 usages  manuu1999
private void performSearch(String queryStr) {
    try {
        FSDirectory directory = FSDirectory.open(Paths.get(INDEX_DIR));
        DirectoryReader reader = DirectoryReader.open(directory);
        IndexSearcher searcher = new IndexSearcher(reader);
        StandardAnalyzer analyzer = new StandardAnalyzer();

        String[] fields = {"title", "genre", "year", "rating"};
        MultiFieldQueryParser parser = new MultiFieldQueryParser(fields, analyzer);
        parser.setAllowLeadingWildcard(true);

        Query query = parser.parse(queryStr);
        TopDocs results = searcher.search(query,  n: 100);

        lastResults.clear();
        for (ScoreDoc hit : results.scoreDocs) {
            lastResults.add(searcher.doc(hit.doc));
        }

        paginator.setupPagination(lastResults);
        paginator.displayCurrentPage();

        reader.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

Edit | Explain | Test | Document | Fix
1 usage  manuu1999
public void filterResults(String filterField, String filterValue) {
    List<Document> filteredResults = new ArrayList<>();
    for (Document doc : lastResults) {
        if (doc.get(filterField).toLowerCase().contains(filterValue.toLowerCase())) {
            filteredResults.add(doc);
        }
    }
    lastResults = filteredResults;
    paginator.setupPagination(lastResults);
    paginator.displayCurrentPage();
}
```

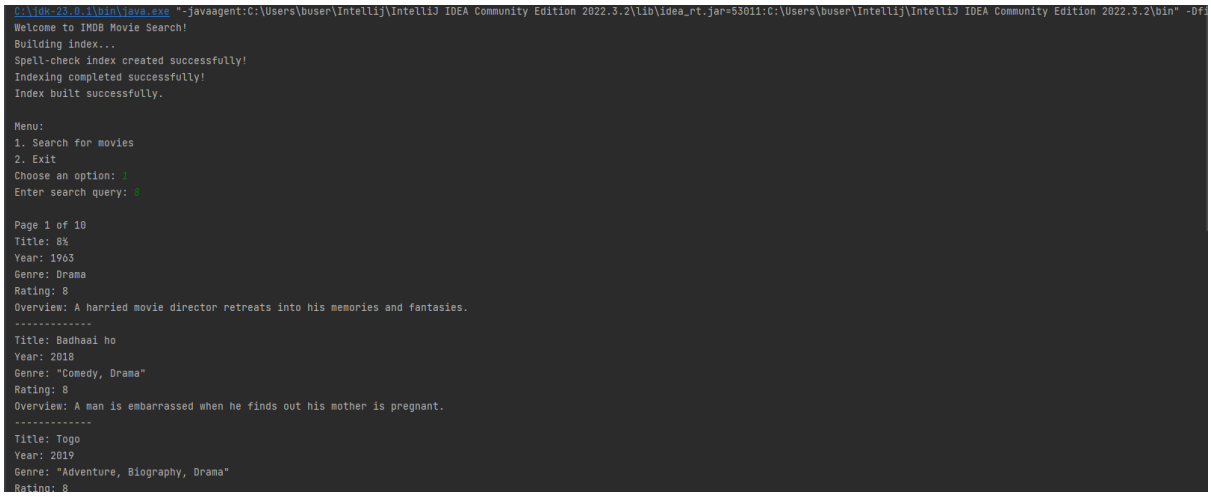


```
Edit | Explain | Test | Document | Fix
2 usages  manuu1999
public void paginate(String direction) {
    if ("next".equalsIgnoreCase(direction)) {
        paginator.nextPage();
    } else if ("prev".equalsIgnoreCase(direction)) {
        paginator.previousPage();
    }
}

1 usage  manuu1999
private boolean isNumeric(String str) {
    return str.matches( regex: "-?\\d+(\\.\\d+)?");
}
}
```

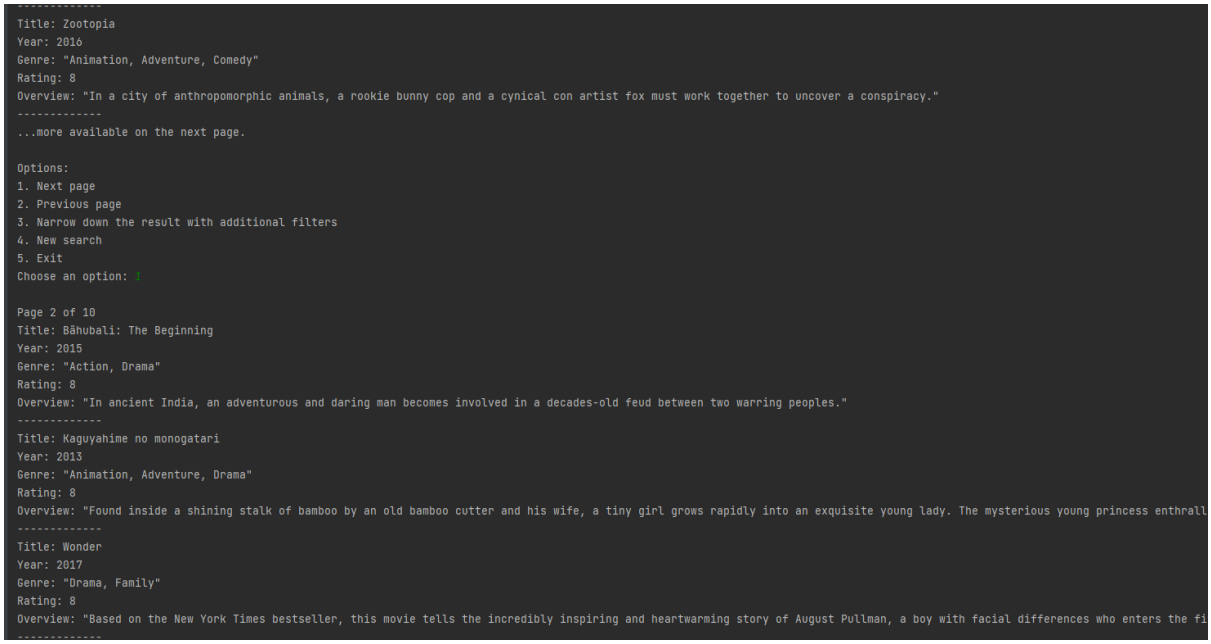
The spell checking didn't make sense for the rating and release year search, so I restricted it to non-numeric values.

Output:

It shows maximum 10 movies at the time but with the commands 1 and 2 you can go further and back in the list:



Go further:



Go back:

With the page x of x in the beginning you can always see where you currently are in the whole search result list.