Assignment 3: KWIC Search

Code Repository URL:	https://github.com/TIC3001/tic3001-ay2122s2-13/tree/assignment3		
Team number:	13		
Name:	Roshan Kumar	Name:	Noel Lim Xian
Student Number:	A0094621H	Student Number:	A0211270Y

1. Introduction

This assignment extends the Pipe & Filter design in Assignment 2 to include MVC design pattern via the implementation of classes Main and Librarian, that handle the storage of data and perform key functions such as search and format.

We categorize the requirements into 2 routines as shown below:

Routine 1. Data Loading

a) Read filenames from ListOfFiles.in file.
b) Parse filenames as a list of Manual filenames.
c) Read titles from each Manual file.
d) Parse titles as an array of lines, with each line an array of words.
e) Concordance is populated from titles for each Manual (Circular shifting).
f) Store the concordance of the Manuals.

This broad routine reads the filenames, parses the titles, performs data processing (Concordance / Shifting etc.) and stores the data into a class called Manuals.

Routine 2. Search Feature

a) Accepts user queries.
b) Execute query and store result.
c) Retrieve and format history of all query results as standard output data.
d) Display output.

* Program loops until user inputs exit keyword.

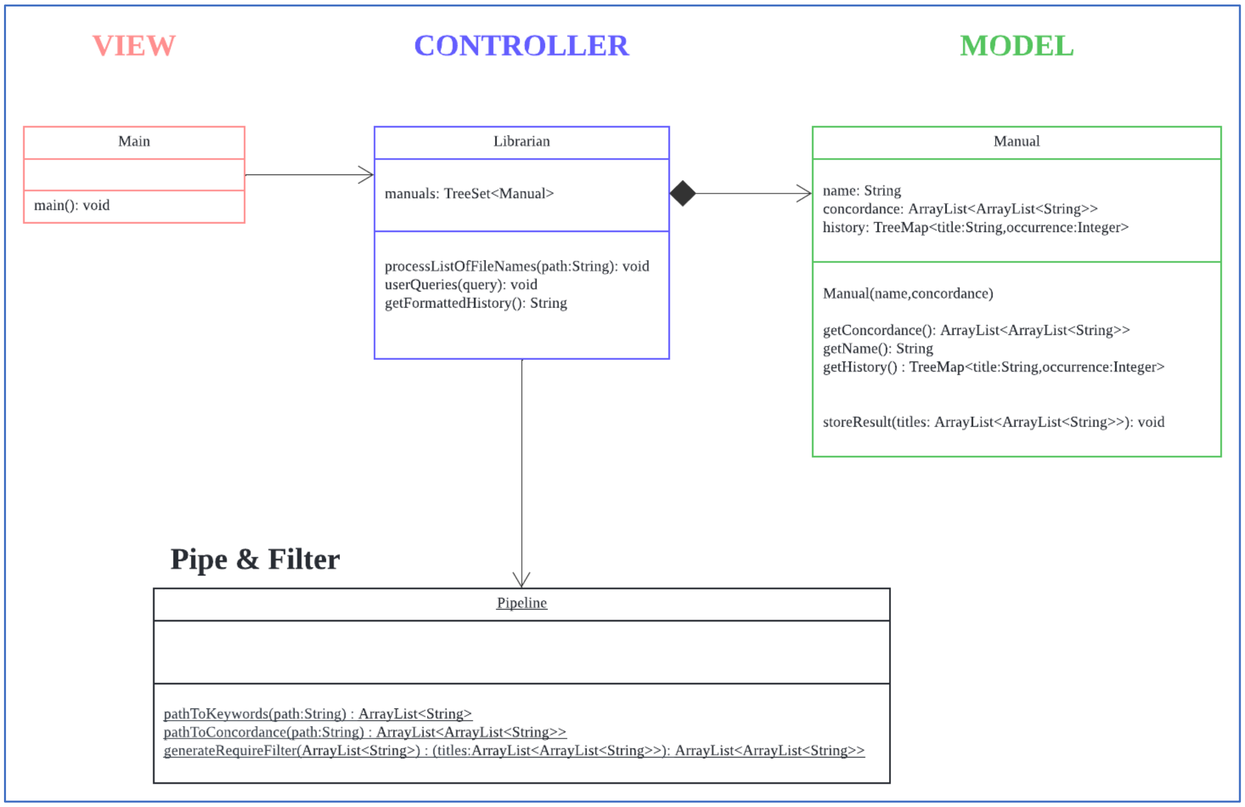
Contributions

Noel & Roshan – Worked on the additional features together using Code with Me – JetBrains during the live session and figured out on the low-level and high-level design implementation concurrently. Candidate set of suitable principles and architectural patterns to adhere to.

2. Architectural Design

Diagrammatic Illustrations

Class Diagram (Overall)


VIEW
CONTROLLER
MODEL
Main
main(): void
Librarian
manuals: TreeSet<Manual>
processListOfFileNames(path:String): void
userQueries(query): void
getFormattedHistory(): String
Manual
name: String
concordance: ArrayList<ArrayList<String>>
history: TreeMap<title:String,occurrence:Integer>
Manual(name,concordance)
getConcordance(): ArrayList<ArrayList<String>>
getName(): String
getHistory() : TreeMap<title:String,occurrence:Integer>
storeResult(titles: ArrayList<ArrayList<String>>): void
Pipe & Filter
Pipeline
pathToKeywords(path:String) : ArrayList<String>
pathToConcordance(path:String) : ArrayList<ArrayList<String>>
generateRequireFilter(ArrayList<String>) : (titles:ArrayList<ArrayList<String>>): ArrayList<ArrayList<String>>

We introduce the class diagram for the overall application and the motivations will be evaluated in the “Design and Principles“ Section.

Main: To trigger the program routine.

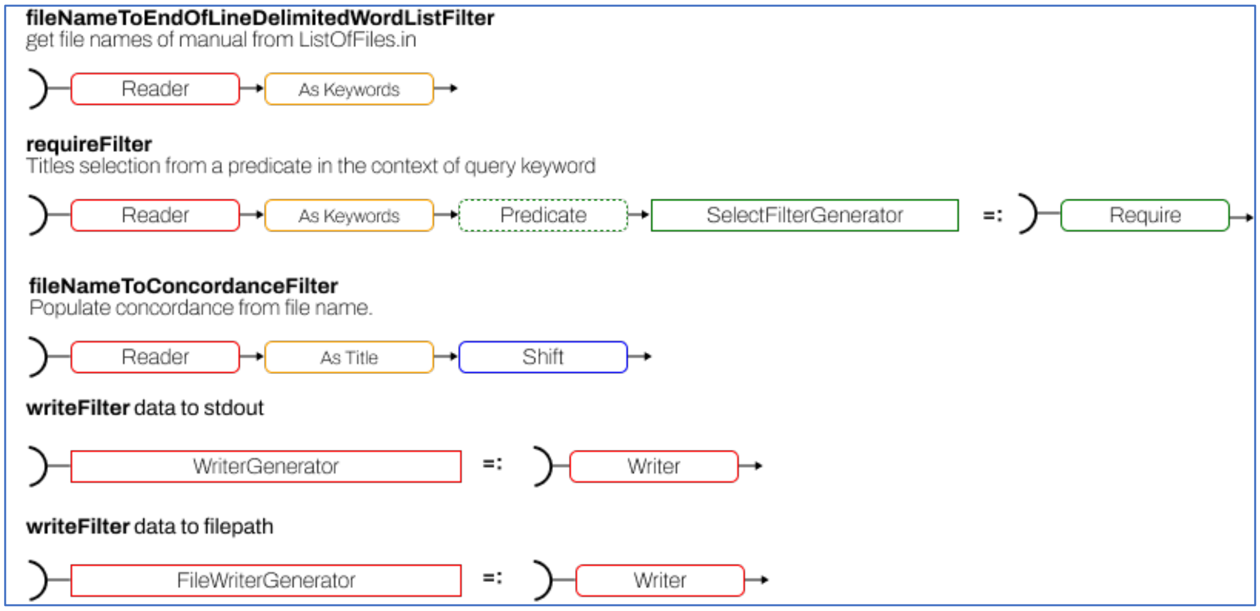
Librarian: Objective: Perform some data processing utilizing Pipe & Filter class and store data to Manual. Also, interact with Main to produce formatted data as output.

Manual: Objective: Store name, concordance, and query result of Manuals, which includes.

Pipeline: Generate output given an input. Output can be a function or value.

On Pipelines and Filter Components

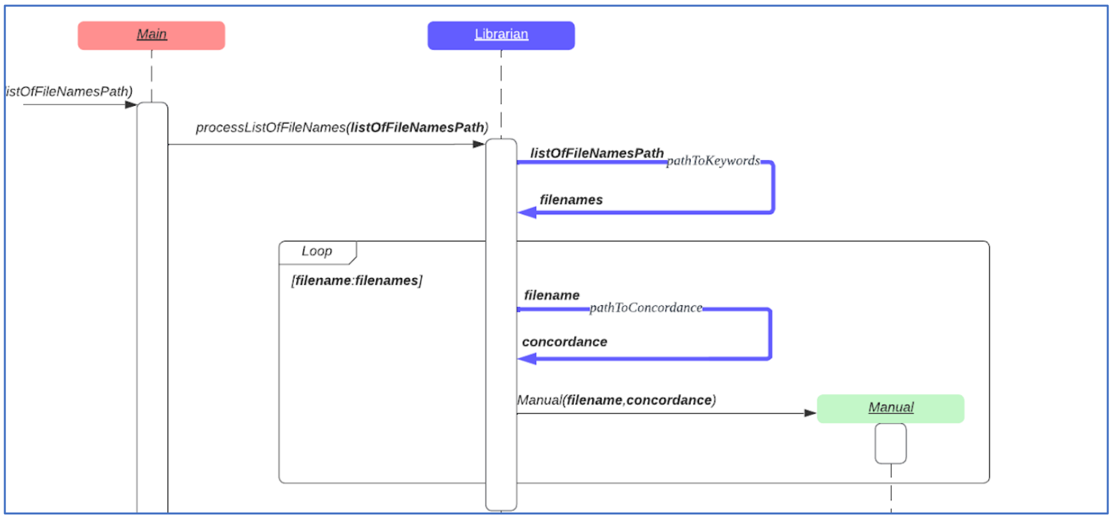
Examples of Pipelines (Component Diagram)


fileNameToEndOfLineDelimitedWordListFilter
get file names of manual from ListOfFiles.in
Reader
As Keywords
requireFilter
Titles selection from a predicate in the context of query keyword
Reader
As Keywords
Predicate
SelectFilterGenerator
=:
Require
fileNameToConcordanceFilter
Populate concordance from file name.
Reader
As Title
Shift
writeFilter data to stdout
WriterGenerator
=:
Writer
writeFilter data to filepath
FileWriterGenerator
=:
Writer

The Pipeline static class reuses components from Assignment 1 & 2 (E.g., Reader / Shift etc.). Pipelines are static and pure methods composed of specific components with sequence of execution defined by the use case. Utilizing the Pipe and Filter architecture and modular nature of filter components is advantageous to due to its flexibility to build custom pipelines that can be shared across the program. It also helps to identify input order of dependencies (refer to Appendix for details).

SEQUENCE DIAGRAM

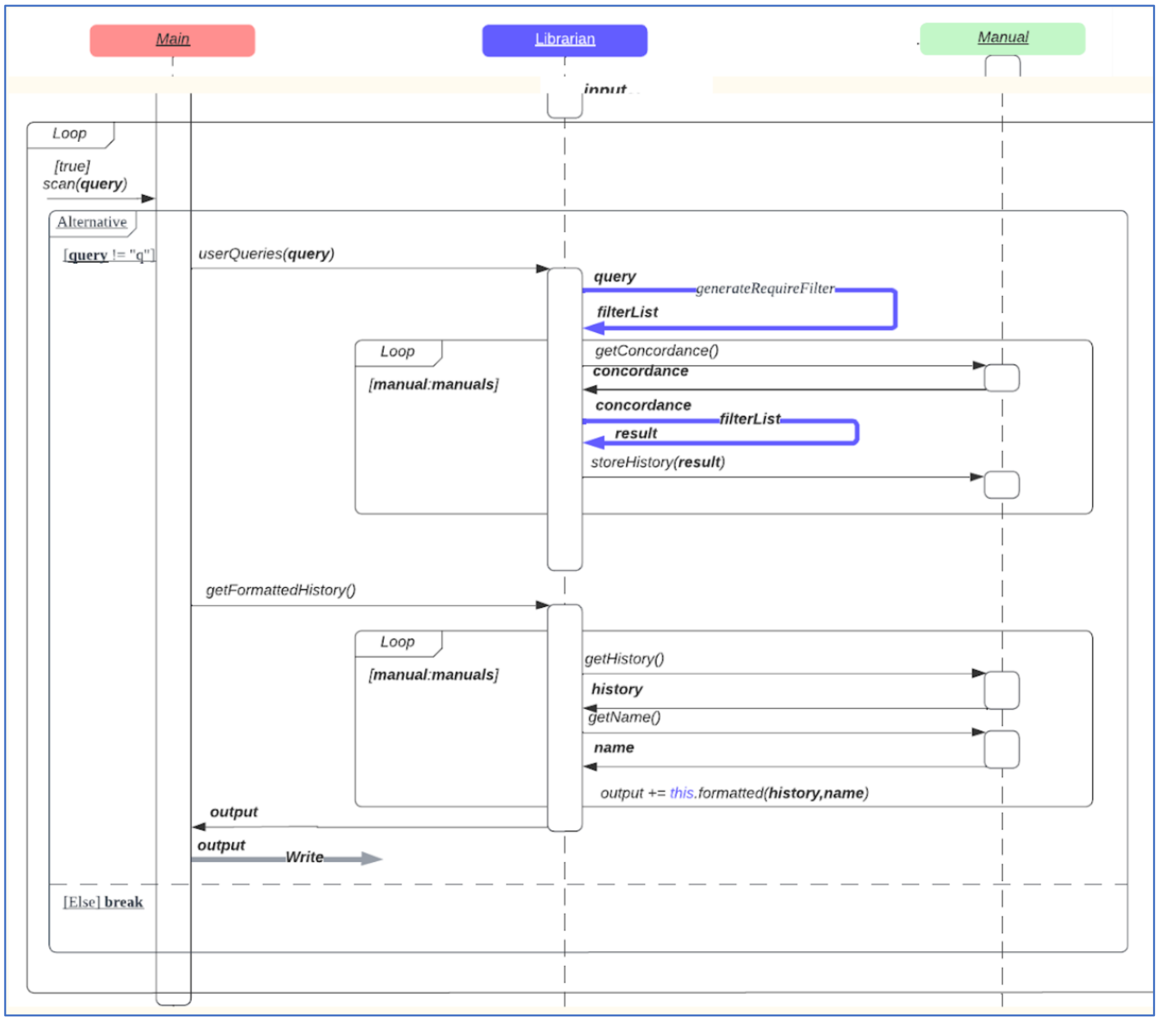
Routine 1 – Data Loading



Once Main has been provided with the filename of the ListofFiles, it notifies the Librarian. The Librarian will run a list of actions to be performed for the data storing:

1) Parse content in *listOfFileNamesPath* as array of all filenames.
2) For each filename, parse content as array of titles, pass the titles to a pipeline to generate concordance. A manual will be initialized, and the name and concordance will be stored in the instance. The manual will then be added into the possession of the Librarian.

Routine 2 – Search



Once data loading is completed, the application is ready to accept user queries. It enters a loop and provides search output results before awaiting for the next user input. The program exits with a message when user input is entered as "q".

When a user query has been registered, the following occurs:

1) Main notifies Librarian to process query input
2) Librarian takes in the query and generates a selection filter ("filterList") using the a filter generator pipeline.

3) For each Manual in the Librarian's possession, it retrieves the concordance and obtain the selected lines of the concordance via the selection filter. The Librarian will record the occurrence of the selected lines in the Manual. It is via this process we can determine which concordance have previous or current match to user queries.
4) Main will request for formatted history of all query results from the Librarian.
5) For each Manual with positive (non-empty) history, the Librarian retrieves and format the history and returns a combined output string.

3. Applying Design Principles and Architectural Patterns

The class components in the program correspond mindfully to the architectural patterns we have engineered. We employed MVC as the main pattern for application service and delegated data

Main (View) - Accepts input and display output

Librarian (Controller) - Data interaction and data (string) formatting

Pipeline (Pipe & Filter) – Data generation and transformation

Manual (Model) - Data storage

processing functions to Pipe & Filter pattern. We will elaborate on the architectural roles and summarize the implications of architectural decisions on design principles of each class.

MVC

First, the Main class is an I/O channeling and application control apparatus. It **lacks awareness** of the application's functional capabilities and has no jurisdiction on how the application should manage and operate data. Its responsibility is to enable visual interaction with the user. The **concerns** of user input and output are also distinct to this class.

Design Principles: **Abstraction**, **Separation of Concern**

Second, the Librarian class contains data and methods that modify application state and implements the applications' logical requirements (recording manuals and formatting data). Evidently, methods in this class are imperative and including instructional steps in one hierarchical level within this class's method promotes **sequential cohesiveness**. Methods for feature extensions (editing and deleting of data) should also be in custody of this class to promote **logical cohesiveness**.

Design Principles: **Sequential Cohesiveness**, **Logical Cohesiveness**

Third, the Manual class is the model representation of a concordance. This allows us to mimic real-world activities such as recording and retrieval of title entries. It can also store relevant information and properties, with persistence, in the manual for reference (occurrence of a title etc.). The class structure of the manuals is hidden from the other classes. To retrieve data, the Manual class will **expose public methods to interface with other classes**.

Design Principles: **Information Hiding**

Pipe & Filter

Fourth, the Pipeline class methods generates output. It is strongly associated with the Librarian as logical operations frequently require data transformation. The complexity of formulating pipelines is delegated to this class and hidden from the caller. Filter components are highly **re-usable** among constructed pipelines. It also adheres to **the Open-Closed Principle** and feature-extend instead of replacing pipelines. For example, if we change the line selection process from one to multiple conditions, we can define a new pipeline by appending (chaining) additional selection filters to the existing pipeline.

Design Principles: **Re-usability, Open-Closed Principle**

4. Search on KWIC

On Paragraph

The stretch requirement is to allow the program to read paragraphs. We define a sentence as a line with the final word appended with a full stop, and sentences in a paragraph are delimited by a space.

Example 1 of a Paragraph with 2 Sentences in quotes:

```
"I am a dog. kawazu tobikomu dog.\n"
```

To generate a manual with paragraphs, we need to pre-process the raw source text by replacing full-stop and space with full-stop and end of line.

Example 2 when Example 1 is pre-processed:

```
"I am a dog.\nkawazu tobikomu dog.\n"
```

Now, we can split the sentences into lines as per normal:

The lines are:

```
"I am a dog."
"kawazu tobikomu dog.
```

For searching, we adjusted the case to reduce the first word of a shifted line to a general case - its form without full-stop at the end.

```
* We consider the case that the line may be shifted, and the original line ended with a full stop.
* That is, the first word of a target line may end with full stop. We may then strip the full stop before applying the predicate function.
* <p>
* Example of filtering a shifted line:
* Original Line : "I am a dog."
* Decomposed: <I,am,a,dog.>
* Shifted: <dog.,I,am,a> => first word is <dog.>
* Before Applying Predicate: first word is converted to <dog>
*
 * @param requireSet set of words that intersects with the first word of a line.
 * @return predicate
 */
public static Predicate newRequiredLineByFirstWordPredicateAgainstSet(HashSet<String> requireSet) {
    return line ->
            requireSet.size() == 0 ||
(requireSet.contains(line.get(0).replaceAll("\\.$",
"").toLowerCase(Locale.ROOT)));
}
```

On Fast Retrieval (Non-functional Requirement)

We maintained a linear time complexity for adding elements and iterating all manuals of Librarian and lines in result history of Manuals by defining the container structures to be Trees with custom comparators complying to requirements.

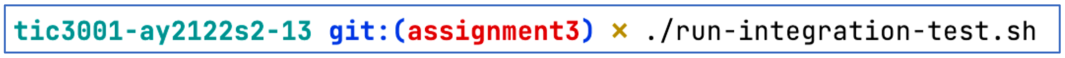
```
public class Librarian {
    /**
     * The insert ordering and aggregate container of manuals are defined
once in the constructor.
     */
    public Librarian() {
        manuals = new TreeSet<>(Comparator.comparing(m ->
m.getName().toLowerCase(Locale.ROOT)));
    }

}

public class Manual {
        private final TreeMap<String, Integer> history = new TreeMap<String,
Integer>(new StringComparator());
        // ...
}
```

5. Testing

On Testing

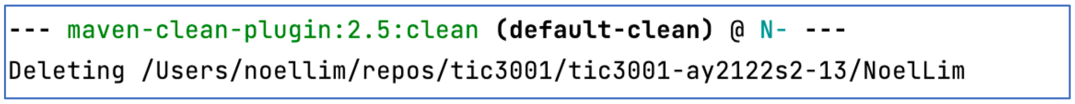
We have replaced the test cases for A2 with A3 and use the same approach use the authoritative integration test script during commits, pushes and merges.

Running integration test….



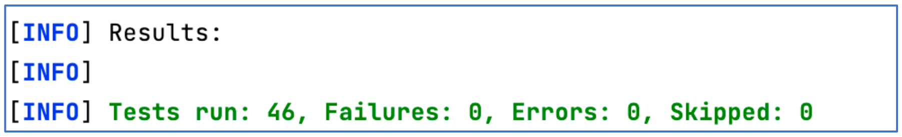
```
tic3001-ay2122s2-13 git:(assignment3) ✗ ./run-integration-test.sh
```

1. Cleans build directory to avoid using stale artefacts.



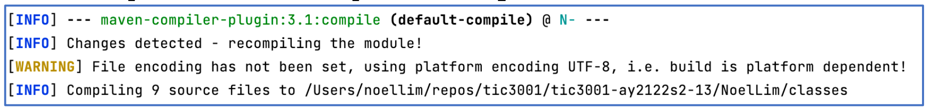
```
--- maven-clean-plugin:2.5:clean (default-clean) @ N- ---
Deleting /Users/noellim/repos/tic3001/tic3001-ay2122s2-13/NoelLim
```

2. Runs source unit test.



```
[INFO] Results:
[INFO]
[INFO] Tests run: 46, Failures: 0, Errors: 0, Skipped: 0
```

3. & 4. Install dependencies Compile to bytecode.



```
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ N- ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 9 source files to /Users/noellim/repos/tic3001/tic3001-ay2122s2-13/NoelLim/classes
```

5. Package into jar.

```
[WARNING] Replacing pre-existing project main-artifact file: /Users/noellim/repos/tic3001/tic3001-ay2122s2-13/NoelLim/classes
with assembly file: /Users/noellim/repos/tic3001/tic3001-ay2122s2-13/NoelLim/KWIC.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

6. System test against jar.

```
running test
Test result: PASSED  1
Test result: PASSED  2
Test result: PASSED  3
Test result: PASSED  4
Test result: PASSED  5
Test result: PASSED  6
Test result: PASSED  7
Test result: PASSED  8
```

End of Report

5. Appendix (Generating Filters using Pipelines)

To prepare a filter that outputs selected lines, the function definitions of individual pipeline stages are as follows:

1) Obtain the required keyword(s):

```java
public static String read(String path) throws IOException {
    try {
        return Files.readString(Path.of(path));
    } catch (NoSuchFileException err) {
        throw new NoSuchFileException("Error opening file " + path + " (No Such
File Exception)");
    }
}
```

2) A predicate function generator consumes the required keyword(s) and produces a filter that accepts a line and outputs a predicate value contingent on the keywords:

```java
/**
 * The predicate interface which applied to a line will return if the line
satisfies some condition.
 */
@FunctionalInterface
interface Predicate {
    boolean shouldSelect(ArrayList<String> line);
}

/**
 * @param keyword The word to compare against first word of target line.
 * @return predicate
 */
public static Predicate
newRequiredLineByFirstWordAgainstSingleKeywordPredicate(String keyword) {
    return line -> line.get(0).replaceAll("\\.$",
"").toLowerCase(Locale.ROOT).equals(keyword.toLowerCase(Locale.ROOT));
}
```

3) A select filter generator consumes the predicate filter and produces a filter that accepts lines and outputs lines which satisfies the predicate function:

```
/**
 * The selection interface which applied to an array lines will return a
subset of selected lines.
 */
@FunctionalInterface
public interface SelectionFilter {
    ArrayList<ArrayList<String>> filter(ArrayList<ArrayList<String>> source);
} /**
 * The selection interface which consumes a predicate function and returns a
{@link SelectionFilter}.
 *
 * @param p
 * @return
 */
public static SelectionFilter newRequireFilter(Predicate p) {
    return lines ->
lines.stream().filter(p::shouldSelect).collect(Collectors.toCollection(ArrayL
ist::new));
}
```