Description for Assignment 3

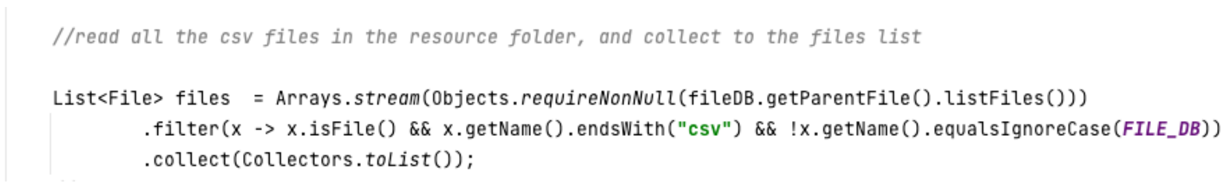
Author: Xiaonan Liu

According to the feedback of last assignment, I did some change on the assignment 2.

Firstly, I add some new packages instead just implements all in default package, only main function is in default package. The new class I use to record the search history is defined in the Model package, and other utils function and class is define in the Tools package.

Secondly, to avoid hard coding the file names, I choose to scan the files in the source, and use lambda expression to filter the files I need to merge, then add those files into the list, then merge them.

Here is the new code for collect the resource file



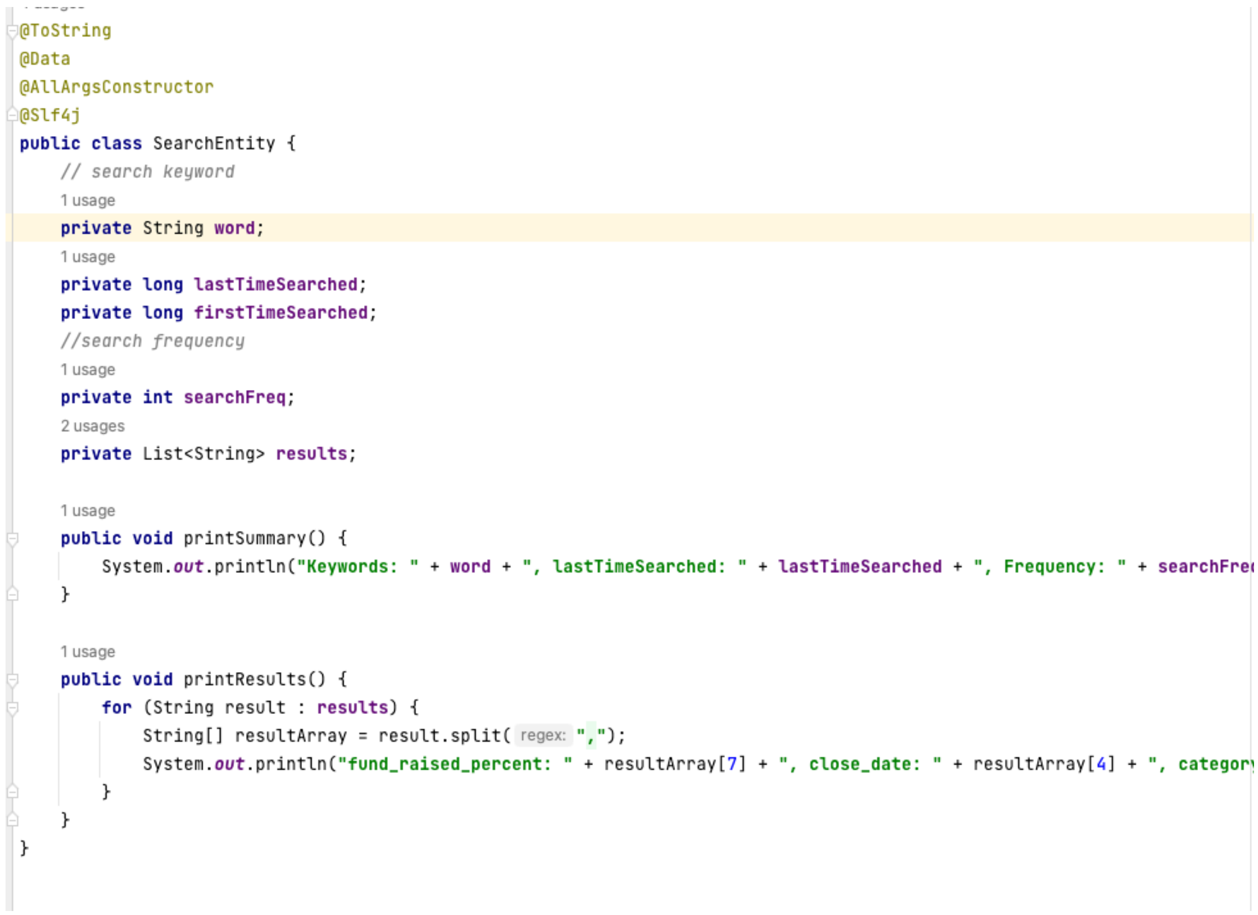
```
//read all the csv files in the resource folder, and collect to the files list

List<File> files  = Arrays.stream(Objects.requireNonNull(fileDB.getParentFile().listFiles()))
        .filter(x -> x.isFile() && x.getName().endsWith("csv") && !x.getName().equalsIgnoreCase(FILE_DB))
        .collect(Collectors.toList());
```

Thirdly, I define the specific version of the Junit to avoid the warning when compile the program.

Also, more tests are provided.

To implement the search memory features, I define a new class in model package, named SearchEntity, this class has five fields, the keyword for searching the first and last time stamp, the search frequency, and the result lists, the Constructors, getters and setters , and toString functions will auto generated by

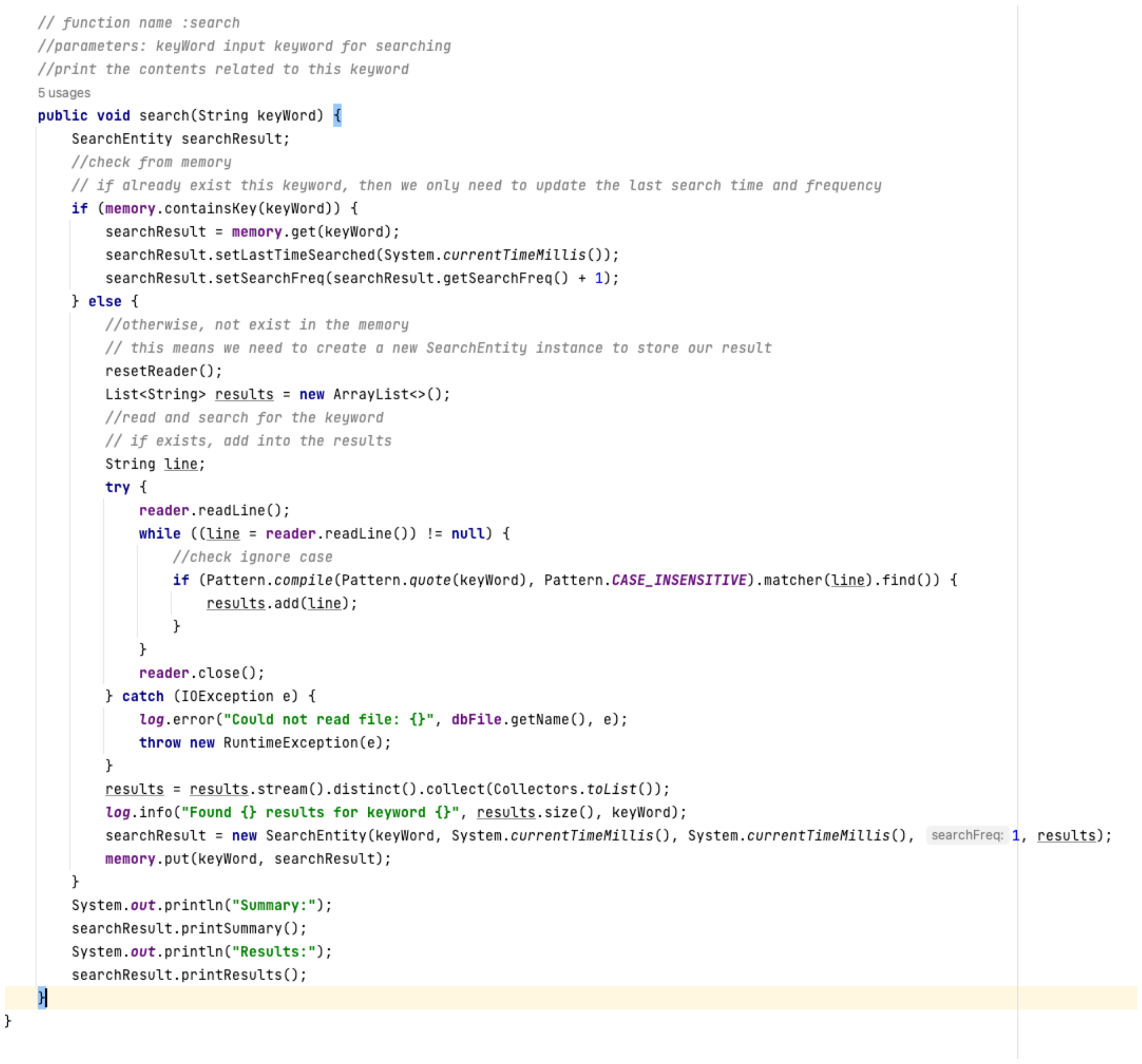
Lombok, I only implement two print functions to simply show the summary and the while result by the format I defined.



```
@ToString
@Data
@AllArgsConstructor
@Slf4j
public class SearchEntity {
    // search keyword
    1 usage
    private String word;
    1 usage
    private long lastTimeSearched;
    private long firstTimeSearched;
    //search frequency
    1 usage
    private int searchFreq;
    2 usages
    private List<String> results;

    1 usage
    public void printSummary() {
        System.out.println("Keywords: " + word + ", lastTimeSearched: " + lastTimeSearched + ", Frequency: " + searchFre
    }

    1 usage
    public void printResults() {
        for (String result : results) {
            String[] resultArray = result.split( regex: ",");
            System.out.println("fund_raised_percent: " + resultArray[7] + ", close_date: " + resultArray[4] + ", categor
        }
    }
}
```

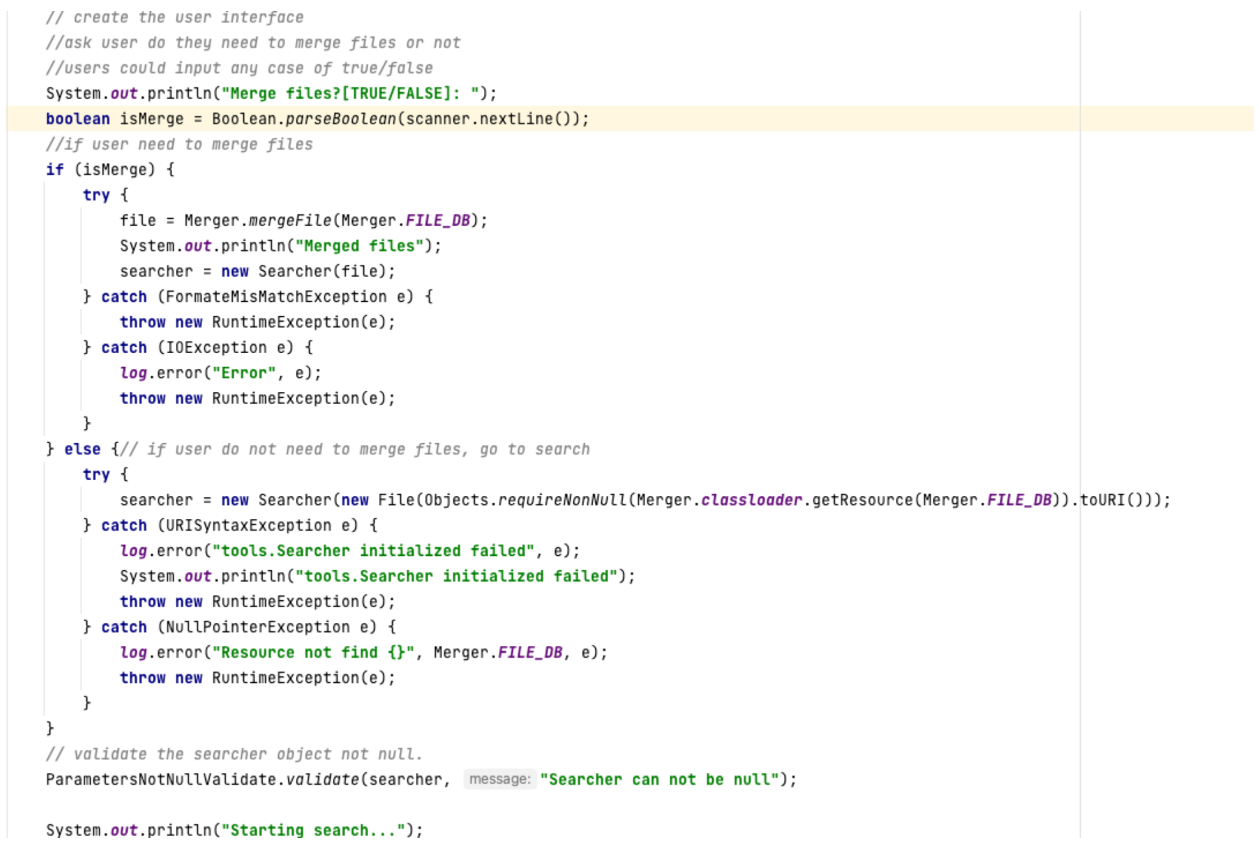
Another important modify are separate merge and search into two individual class, and the memory system are defined in the Searcher, this memory system use the keyword for searching as key, and the new class we just defined as the value.

Also, some necessary updates are added into search function because we need to update the search histories into our memory system.



```java
    // function name :search
    //parameters: keyWord input keyword for searching
    //print the contents related to this keyword
    5 usages
    public void search(String keyWord) {
        SearchEntity searchResult;
        //check from memory
        // if already exist this keyword, then we only need to update the last search time and frequency
        if (memory.containsKey(keyWord)) {
            searchResult = memory.get(keyWord);
            searchResult.setLastTimeSearched(System.currentTimeMillis());
            searchResult.setSearchFreq(searchResult.getSearchFreq() + 1);
        } else {
            //otherwise, not exist in the memory
            // this means we need to create a new SearchEntity instance to store our result
            resetReader();
            List<String> results = new ArrayList<>();
            //read and search for the keyword
            // if exists, add into the results
            String line;
            try {
                reader.readLine();
                while ((line = reader.readLine()) != null) {
                    //check ignore case
                    if (Pattern.compile(Pattern.quote(keyWord), Pattern.CASE_INSENSITIVE).matcher(line).find()) {
                        results.add(line);
                    }
                }
                reader.close();
            } catch (IOException e) {
                log.error("Could not read file: {}", dbFile.getName(), e);
                throw new RuntimeException(e);
            }
            results = results.stream().distinct().collect(Collectors.toList());
            log.info("Found {} results for keyword {}", results.size(), keyWord);
            searchResult = new SearchEntity(keyWord, System.currentTimeMillis(), System.currentTimeMillis(), searchFreq: 1, results);
            memory.put(keyWord, searchResult);
        }
        System.out.println("Summary:");
        searchResult.printSummary();
        System.out.println("Results:");
        searchResult.printResults();
    }
}
```

In the main function, the user interface is create to ask users to input "true/false" (in any case) to determine the command for this system, if user input true, that means they need to merge the file, otherwise the system will switch to search command.


```java
// create the user interface
//ask user do they need to merge files or not
//users could input any case of true/false
System.out.println("Merge files?[TRUE/FALSE]: ");
boolean isMerge = Boolean.parseBoolean(scanner.nextLine());
//if user need to merge files
if (isMerge) {
    try {
        file = Merger.mergeFile(Merger.FILE_DB);
        System.out.println("Merged files");
        searcher = new Searcher(file);
    } catch (FormateMisMatchException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        log.error("Error", e);
        throw new RuntimeException(e);
    }
} else {// if user do not need to merge files, go to search
    try {
        searcher = new Searcher(new File(Objects.requireNonNull(Merger.classloader.getResource(Merger.FILE_DB)).toURI()));
    } catch (URISyntaxException e) {
        log.error("tools.Searcher initialized failed", e);
        System.out.println("tools.Searcher initialized failed");
        throw new RuntimeException(e);
    } catch (NullPointerException e) {
        log.error("Resource not find {}", Merger.FILE_DB, e);
        throw new RuntimeException(e);
    }
}
// validate the searcher object not null.
ParametersNotNullValidate.validate(searcher, message: "Searcher can not be null");

System.out.println("Starting search...");
```

As long as user does not input the “q/Q”, the system will continue run for merge and search commands.