CSC319 – J3 JAVA LAMBDA AND STREAMS – REVIEW EXERCISE

Due date: 5 May 2022, at NOON on CSCMS
Solution review: 6 May 2022, 1:30PM-3PM on Microsoft Teams*
Instruction: Work the following problems and zip all your answers into one (1) single *.zip* file for submission. This review exercise contributes no scores to your final grade.
**Remark*: Participation in the solution review is NOT required. The review session will be recorded.

Q1. Write a Java application in one (1) single file that modifies the Execute-Around code that we studied in class such that now the **processFile** method will return a list of all words from the file, ***call_of_the_wild.txt***. Once you have the list of all words, you will sort these words in the list in a descending order using ***the number of characters*** as a criterion (for simplicity, *no need to break the tie*). Then you will print out the ***first occurrence of the longest word*** in the list. Additional requirements are as follows:

- Do **NOT** use wildcard to import classes;
- Your application will read a file called ***call_of_the_wild.txt***. The file is provided;
- For simplicity, assume the file name is fixed. Do **NOT** prompt the user for the file name;
- Empty string and spaces must NOT be treated as a word;
- Your application must include at least one (1) lambda and one (1) method reference;
- Your application **MUST NOT** use Java Streams.

Hint: The **reversed()** method on a lambda expression can be useful for this question.

Q2. Consider the code below, where you can find definitions of the related classes and variables in the accompanied ***.zip*** file. You task in this question is to refactor the given code using the Streams API and lambda expressions. Write a Java application to verify that your refactored code works correctly. Here is the additional requirements:

- You must write one (1) Java file only. This file should compile together with the provided Java code to produce the executable Java application;
- **Extra credits (2 pts)**: Prepare this Java file in such a way that both the original code (method) and the refactored code (method) co-exist in the same class, ***without*** having to change any of the method names.

```
// Here is the code to be refactored
    public Set<String> findLongTracks(List<Album> albums) {
        Set<String> trackNames = new HashSet<>();
        for(Album album : albums) {
            for (Track track : album.getTrackList()) {
                if (track.getLength() > 60) {
                    String name = track.getName();
                    trackNames.add(name);
                }
            }
        }
        return trackNames;
    }
```