Hands-On Guide

Written By: Daoud Kabha

Handson – 1

In this session you will:

- **Define search tasks functional interface**
- **Provide a dynamic search platform**
- **Enrich with search utilities referenced methods**

To complete this session follow these instructions:

- Create a Functional interface search8.SearchTask :

SearchTask<T>
+ startSearch() : long + endSearch() : long + search (List<T>) : List<T>

  - Turn both startSearch() & endSearch into default methods
  - Implement both to return current time in millis

- Create search8.SearchEngine class

SearchEngine<T>
- List<T> data - long lastSearchTimeInMillis
+ getData() : List<T> + setData(List<T>) : void + getLastSearchTimeInMillis() : long + executeSearch(SearchTask<T>) : List<T>

  - Implement executeSearch() to
    - execute a search
    - measure search time and store it in lastSearchTimeInMillis

- Create search8.string.StringSearchUtils class

StringSearchUtils
- String toSearch
+ StringSearchUtils(String toSearch) + startsWith(List<String>) : List<String> + endsWith(List<String>) : List<String> + contains(List<String>) : List<String> + topThree(List<String>) : List<String> <<static>>

  - Implement each of the non-static methods to return a sub-list after filtering the origin collection with the 'toSearch' value
  - Implement the static method to return a sub-list containing the 3 first elements taken from the origin collection (you may use List.subList(..))

- Use the given Test class and edit it
  - Test.main is already implemented to load "Hobbit.txt" from your project and load it as a collection of strings (find it at: *'HandsOn'* directory)
  - Drag Hobbit.txt to the root of your project and make sure you can see it in the Eclipse Project Explorer (find it at: *'HandsOn'* directory)
  - You may run Test just to make sure that the file can be found and loaded (if no IOException is thrown you're fine)
  - Complete Test.main to do the following:
    - Instantiate a SearchEngine<String> and populate it with Hobbit data
    - Execute dynamic search to print all words longer than 5 letters
    - Print the time of the search execution
    - Instantiate StringSearchUtils with the value "a" to search
    - Execute via method reference - contains, startsWith, endsWith and print results
    - Print via static method reference the top three

- Examine the outcomes

HandsOn – 2

In this session you will:

- **Use streams collective operations**
- **Examine streams & parallel streams performance**

To complete this session follow these instructions:

- Use the given Test2 class and edit it
  - Test2.main is already implemented to load "Hobbit.txt" from your project and load it as a collection of strings (find it at: *'HandsOn'* directory)
  - Obtain a stream from hobbit words collection and perform the following:
    - Print word count
    - Print unique word count
    - Print the first word using stream API
    - Sort and then print the first word using stream API
    - Sort in an opposite order and print the first word using stream API
      - In this case apply Comparator dynamically to the Stream<T>.sorted(Comparator<T>) method
  - Test already measures time of execution and prints it by the end of main. Make sure to place you code in the right place so it will be counted as well
  - Now, do the same using parallel streams
  - Do that by <u>replacing</u> streams with parallel streams – don't copy and update
    - Note: appending the two scenarios will not provide real results since first time code is executed it is loaded, interpreted and cached – so repeating the same operation will always execute faster (with no regards of parallelism). Therefore, code must be tested in two different compilations: one that uses streams and another version that uses parallel streams.

HandsOn – 3

In this session you will:

- **Use streams matching operations**
- **Use filter, map and for-each**

To complete this session follow these instructions:

- Edit StringSearchUtil class
  - Provide 3 additional Predicate<String> like methods to reference later:
    - startsWith(String) : boolean
    - endsWith(String) : boolean
    - contains(String) : boolean
    - equals(String) : boolean
- Use the given Test3 class and edit it
  - Test3.main is already implemented to load "Hobbit.txt" from your project and load it as a collection of strings (find it at: *'HandsOn'* directory)
  - Obtain a stream from hobbit words collection
  - Instantiate StringSearchUtil with the following to search value: "aba" and use Stream.anyMatch() on each of the 4 new methods by referencing
  - Use Stream.map() to show a distinct list of all word's length
  - Use Stream.forEach() to print the whole store while:
    - Performing 'break row' (new line) every 20 words
    - Transforming each lower case single 'i' into 'I'
    - Note: if you are planning on using counter – keep in mind that dynamically executed code has its own independent scope much like anonymous classes. Since references of methods and anonymous classes can live outside of their creating stack, they cannot use non-final local variables…

HandsOn – 4

In this session you will:

- **Use streams collectors**
- **Use groupingBy and partitionBy**
- **Use reduce on streams**

To complete this session follow these instructions:

- Use the given Test4 class and edit it
  - Test4.main is already implemented to load "Hobbit.txt" from your project and load it as a collection of strings (find it at: *'HandsOn'* directory)
  - Obtain a stream from hobbit words collection
  - Use Collectors to present:
    - sum of all letters in the text
    - total length statistics (sum, avg, min, max…)
    - average word length
  - Use groupingBy to present:
    - A map contains:
      - Key = words first letter
      - Value = collection of all words beginning with the key
      - Make sure to work on a distinct stream
    - A map contains:
      - Key = word first letter
      - Value = no' of occurrences of words beginning with the key in the text
  - Use partitioning to present:
    - All words without 'a','e','I' and 'u'
  - Use reduce to present:
    - Longest word in the text (first found)
    - Shortest word in the text (first found)