**Java Journal Template**

**Directions:** Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

**Name: Nickolus H olt-Shirley**

**Date:6/20/23**

**Final Replit Program Share Link:**

Complete the following template. Fill out all entries using complete sentences.

## PART 1: Defining Your Problem

|  |
| --- |
| **Task**  State the problem you are planning to solve.  **Requirements**   * Describe any input data you expect to use. * Describe what the program will do to solve the problem. * Describe any outputs or results the program will provide.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Why do you want to solve this particular problem? * What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source? * Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue? * What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete? |
| Problem: Sorting Entries based on different criteria  Requirements:  - Input data: The program expects a file named "input.txt" that contains entries in the format of name-count pairs, separated by a comma. The number of entries to be read is specified by the variable `numEntries` in the code.  - Program solution: The program will read the entries from the input file, sort them based on the specified mode (name, count, nameThenCount, countThenName), using the appropriate comparator. The sorting algorithm used is merge sort. The sorted entries will be stored in an array.  - Outputs: The program will write the sorted entries to a file named "output.txt". It will also print a message indicating the completion of the sorting process.  Inspiration:  - The problem is to sort a collection of entries based on different criteria, which can be useful in various scenarios such as data analysis, statistics, or organizing information.  - The source of data is an external file, "input.txt," which allows the program to work with different datasets without modifying the code.  - There is no need for user interaction throughout the program. The input data is read from the file, and the sorted results are written to another file. The user is informed of the completion of the sorting process through a printed message.  - The expected result is a file named "output.txt" containing the sorted entries. The program will print a message indicating that the sorting process is completed, along with the file name where the sorted data is stored. |

## PART 2: Working Through Specific Examples

|  |
| --- |
| **Task**  Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.  **Requirements**  Complete the three steps below **for at least two distinct examples/scenarios**.   * State any necessary input data for your simplified problem. * Write clear and specific steps in English (not Java) detailing what the program will do to solve the problem. * Describe the specific result of your example/scenario.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Are there any steps that you don’t fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots. * Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don’t understand. Rewrite these parts as clearly as you can. * Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work? |
| Example 1: Sort Entries by Name in Ascending Order  Input data:  - File: "input.txt" containing entries in the format of name-count pairs (e.g., "John, 5", "Alice, 3", "Bob, 7").  Steps to solve the problem:  1. Read the entries from the "input.txt" file.  2. Create an instance of the `Sorter` class.  3. Retrieve the comparator for sorting by name.  4. Perform the merge sort algorithm on the entries using the name comparator, sorting them in ascending order by name.  5. Write the sorted entries to a file named "output.txt".  6. Print a message indicating the completion of the sorting process.    Result:  - The program will generate a file named "output.txt" containing the entries sorted in ascending order based on the name. For example:  Input.txt file contains  Daniel,5 Cathy,5 Bob,5 Adam,5 Daniel,4 Cathy,4 Bob,4 Adam,4 Daniel,3 Cathy,3 Bob,3 Adam,3 Daniel,2 Cathy,2 Bob,2 Adam,2 Daniel,1 Cathy,1 Bob,1 Adam,1  And the output after compiling  Output.txt file  Adam,1 Adam,2 Adam,3 Adam,4 Adam,5 Bob,1 Bob,2 Bob,3 Bob,4 Bob,5 Cathy,1 Cathy,2 Cathy,3 Cathy,4 Cathy,5 Daniel,1 Daniel,2 Daniel,3 Daniel,4 Daniel,5    NB// the input file "input.txt" should contain valid 20 entries in the specified format. |

## PART 3: Generalizing Into Pseudocode

|  |
| --- |
| **Task**  Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.  **Requirements**   * Write pseudocode for the program in English but refer to Java program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.   **Help With Writing Pseudocode**   * Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it, other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert to code later.   + <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>   + <https://www.wikihow.com/Write-Pseudocode>   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well. * Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them. * When you are finished with your pseudocode, does it make sense, even to a person that does not know Java? Aim for the clearest description of the steps, as this will make it easier to convert into program code later. |
| 1. Define the Entry class with name and count as private variables.  2. Implement constructors and accessor methods for the Entry class.  3. Override the toString() method in the Entry class to provide a string representation of an Entry object.  4. Implement the static method fromFile() in the Entry class to read entries from a file and return an array of Entry objects.  5. Implement the static method toFile() in the Entry class to write entries to a file.  6. Define the NameComparator class that implements the Comparator interface and compares entries based on name in a case-insensitive manner.  7. Define the CountComparator class that implements the Comparator interface and compares entries based on count.  8. Define the NameThenCountComparator class that implements the Comparator interface and compares entries based on name first, and if the names are equal, it compares based on count.  9. Define the CountThenNameComparator class that implements the Comparator interface and compares entries based on count first, and if the counts are equal, it compares based on name.  10. Define the Sorter class.  11. Implement the merge() method in the Sorter class to merge two sorted subarrays during the merge sort process.  12. Implement the mergeSort() method in the Sorter class to perform the merge sort algorithm recursively.  13. Implement the getComparator() method in the Sorter class to return the appropriate comparator based on the specified mode.  14. In the main() method:  - Define the inputFileName, numEntries, mode, and outputFileName variables.  - Call the fromFile() method in the Entry class to read entries from the input file and store them in an array.  - Create an instance of the Sorter class.  - Call the getComparator() method in the Sorter class to get the appropriate comparator based on the mode.  - Call the mergeSort() method in the Sorter class to sort the entries using the comparator.  - Call the toFile() method in the Entry class to write the sorted entries to the output file.  - Print a message indicating the completion of the sorting process. |

## PART 4: Testing Your Program

|  |
| --- |
| **Task**  While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.  **Requirements**   * For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.   For each error that occurs while writing and testing your code:   * Record the details of the error from Replit. A screenshot or copy-and-paste of the text into the journal entry is acceptable. * Describe what you attempted in order to fix the error. Clearly identify which approach was the one that worked.   **Inspiration**  When writing your entry below, ask yourself the following questions:   * Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program. * Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa? * Did the outcome look the way you expected? Was it formatted correctly? * Does your output align with the solution to the problem you coded for? |
| Test Case 1: input test entries  - Description: Test the program with a valid input file containing 20 entries.  - Steps:  1. Create an "input.txt" file with valid entries (name-count pairs and 10 entries).  2. Set the appropriate input file name, number of entries, sorting mode, and output file name in the code.  3. Run the program.  4. Verify that the program completes without any exceptions or errors.      When encountering such error, rectify the entries must be 20.  Test Case 2: Invalid Input File Format  - Description: Test the program with an input file containing entries in an invalid format (e.g., missing count or comma).  - Steps:  1. Create an "input.txt" file with entries in an incorrect format (missing count or comma).  2. Set the appropriate input file name, number of entries, sorting mode, and output file name in the code.  3. Run the program.  4. Expect an exception or error to be thrown during the execution due to the incorrect input file format.  5. Capture the error details and analyze the error message to identify the cause of the issue.  6. Modify the input file to correct the format, considering the required name-count pairs separated by commas.  7. Repeat the steps from Test Case 1 to validate the program's functionality.  Error: Invalid input file format exception.  - Error Details: The program throws an exception due to an incorrect format of the input file entries.  - Approach to Fix:  - Analyze the error message to identify the cause of the exception (e.g., missing count or comma).  - Review the code that reads and parses the input file (`fromFile()` method) to ensure it correctly handles the expected format.  - Update the code to handle different input file formats and provide appropriate error handling or feedback to the user.  - Retry the test case with a corrected input file and validate that the program runs without errors. |

## PART 5: Commenting Your Program

|  |
| --- |
| **Task**  Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.  **Requirements**   * The purpose of the program and each of its parts should be clear to a reader that does not know the Java programming language.   **Inspiration**  When writing your entry, you are encouraged to consider the following:   * Is each section or sub-section of your code commented to describe what the code is doing? * Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer. |
| //first in the folder create input.txt file and enter the entries in format of name, count eg peter, 4 . the //entries must be 20, next create output.txt file where output will be written.  import java.io.File;  import java.io.FileNotFoundException;  import java.io.PrintWriter;  import java.util.Scanner;  import java.util.Comparator;  class Entry {  private final String name;  private final Integer count;  public Entry(String name, Integer count) {  this.name = name;  this.count = count;  }  // Getters for name and count  public String getName() {  return name;  }  public Integer getCount() {  return count;  }  // Returns a formatted string representation of the Entry object  public String toString() {  return String.format("%s,%d", this.name, this.count);  }  // Reads entries from a file and returns an array of Entry objects  public static Entry[] fromFile(String fileName, int numEntries)  throws IllegalArgumentException, FileNotFoundException {  Entry[] entries = new Entry[numEntries];  try (Scanner scanner = new Scanner(new File(fileName))) {  int index = 0;  while (scanner.hasNextLine() && index < numEntries) {  String line = scanner.nextLine();  String[] parts = line.split(",");  String name = parts[0].trim();  int count = Integer.parseInt(parts[1].trim());  entries[index] = new Entry(name, count);  index++;  }  }  return entries;  }  // Writes the entries to a file  public static void toFile(String fileName, Entry[] entries, int numEntries)  throws IllegalArgumentException, FileNotFoundException {  try (PrintWriter writer = new PrintWriter(fileName)) {  for (int i = 0; i < numEntries; i++) {  writer.println(entries[i].toString());  }  }  }  }  // Comparator for comparing entries based on name  class NameComparator implements Comparator<Entry> {  public int compare(Entry entry1, Entry entry2) {  return entry1.getName().compareToIgnoreCase(entry2.getName());  }  }  // Comparator for comparing entries based on count  class CountComparator implements Comparator<Entry> {  public int compare(Entry entry1, Entry entry2) {  return entry1.getCount().compareTo(entry2.getCount());  }  }  // Comparator for comparing entries based on name, then count  class NameThenCountComparator implements Comparator<Entry> {  public int compare(Entry entry1, Entry entry2) {  int nameComparison = entry1.getName().compareToIgnoreCase(entry2.getName());  if (nameComparison != 0) {  return nameComparison;  }  return entry1.getCount().compareTo(entry2.getCount());  }  }  // Comparator for comparing entries based on count, then name  class CountThenNameComparator implements Comparator<Entry> {  public int compare(Entry entry1, Entry entry2) {  int countComparison = entry1.getCount().compareTo(entry2.getCount());  if (countComparison != 0) {  return countComparison;  }  return entry1.getName().compareToIgnoreCase(entry2.getName());  }  }  // Sorter class for performing merge sort on Entry objects  public class Sorter {  // Merge function for merging two sorted subarrays  private void merge(Entry[] items, int left, int middle, int right, Comparator<Entry> comparator) {  int n1 = middle - left + 1;  int n2 = right - middle;  Entry[] leftArr = new Entry[n1];  Entry[] rightArr = new Entry[n2];  System.arraycopy(items, left, leftArr, 0, n1);  System.arraycopy(items, middle + 1, rightArr, 0, n2);  int i = 0, j = 0;  int k = left;  while (i < n1 && j < n2) {  if (comparator.compare(leftArr[i], rightArr[j]) <= 0) {  items[k] = leftArr[i];  i++;  } else {  items[k] = rightArr[j];  j++;  }  k++;  }  // Copy remaining elements from the left subarray, if any  while (i < n1) {  items[k] = leftArr[i];  i++;  k++;  }  // Copy remaining elements from the right subarray, if any  while (j < n2) {  items[k] = rightArr[j];  j++;  k++;  }  }  // Merge sort function for sorting an array of Entry objects  private void mergeSort(Entry[] items, int left, int right, Comparator<Entry> comparator) {  if (left < right) {  int middle = left + (right - left) / 2;  // Recursive calls to divide the array into subarrays  mergeSort(items, left, middle, comparator);  mergeSort(items, middle + 1, right, comparator);  // Merge the sorted subarrays  merge(items, left, middle, right, comparator);  }  }  // Returns the appropriate comparator based on the given mode  private Comparator<Entry> getComparator(String mode) throws IllegalArgumentException {  switch (mode) {  case "name":  return new NameThenCountComparator(); // Modified to use NameThenCountComparator  case "count":  return new CountComparator();  case "nameThenCount":  return new NameThenCountComparator();  case "countThenName":  return new CountThenNameComparator();  default:  throw new IllegalArgumentException("Invalid mode: " + mode);  }  }  // Main method  public static void main(String[] args) throws Exception {  String inputFileName = "input.txt"; // File to read entries from  int numEntries = 20; // Number of entries to read  String mode = "name"; // Sorting mode (name, count, nameThenCount, countThenName)  String outputFileName = "output.txt"; // File to write sorted entries to  try {  Entry[] entries = Entry.fromFile(inputFileName, numEntries);  Sorter sorter = new Sorter();  Comparator<Entry> comparator = sorter.getComparator(mode);  // Sort the entries using merge sort  sorter.mergeSort(entries, 0, numEntries - 1, comparator);  // Write the sorted entries to a file  Entry.toFile(outputFileName, entries, numEntries);  System.out.println("Sorting completed. The sorted data is stored in " + outputFileName);  } catch (Exception e) {  System.out.println(e.getMessage());  throw e;  }  }  } |

## PART 6: Your Completed Program

|  |
| --- |
| **Task**  Provide the Replit link to your full program code.  **Requirements**   * The program must work correctly with all the comments included in the program.   **Inspiration**   * Check before submitting your Touchstone that your final version of the program is running successfully. |
| <https://github.com/Nikolus93/Nick.git> |