**Exercise 6 – Most Popular Names**

**1. Introduction**

This Java program processes and analyzes the popularity of names based on gender. It leverages HashMap data structures to efficiently store name occurrences, sorts the names by frequency, allows user searches, and merges the data for male and female names into a single list. The sorted data is then written to a file, with the top 200 names being recorded.

**2. Key Components of the Code**

**a. Imports and Libraries**

The code begins by importing several Java classes required for file handling, user input, and data processing:

* **BufferedReader** and **BufferedWriter**: These are used for reading from and writing to files.
* **FileReader** and **FileWriter**: Handle file input and output operations.
* **IOException**: Catches any potential input/output exceptions.
* **HashMap** and **Map**: These store and manipulate the name data.
* **List** and **Collectors**: These are used for sorting and collecting the name occurrences.
* **Scanner**: Handles user input.

**b. Main Data Structures**

Two HashMap instances, maleData and femaleData, are used to store names as keys and their occurrence counts as values. The use of HashMap is efficient for this task because it allows constant-time lookups, and merging and sorting the data becomes easier later in the program.

**c. File Reading and Data Population**

The program opens two files: one for male names and one for female names. Each line in these files contains a name and its associated occurrence count, which are extracted using BufferedReader and split using String.split(). These values are then stored in the respective HashMap.

**d. Sorting the Data**

After populating the HashMap with name data, the program sorts both maleData and femaleData in descending order based on the occurrence counts. This is achieved using Java's Stream API, where the Map.Entry objects are sorted using the compareTo() method. This step prepares the data for efficient searching and display.

**e. User Input and Search**

The program prompts the user to input a gender and a name they wish to search for. Based on the gender (Male or Female), the program searches the sorted list and finds the rank of the given name. If the name is found, the rank and number of occurrences are displayed. If the name is not found, the user is informed with a message.

**f. Merging Data from Male and Female Lists**

After processing the user’s search request, the program proceeds to merge the data from both genders into a single HashMap. The merge() method is used to combine the data, summing the occurrences of common names across the two lists. The merged data is then sorted in descending order by frequency.

**g. Writing the Top 200 Names to a File**

The final step of the program writes the top 200 names (or fewer if there are not enough entries) to a file named Top200Names.txt. The program writes the names in a formatted style with their rank and occurrence count. This list is generated by iterating over the sorted merged data and writing the results to the file using BufferedWriter.

**3. Thought Process Behind the Code**

**a. Choice of Data Structures**

The use of HashMap for storing the name data is a deliberate decision for efficiency. The HashMap allows for fast retrieval and updates, which is crucial when dealing with large datasets. This also simplifies the merging of data between male and female names, as the merge() method handles duplicates gracefully.

**b. Sorting and Stream API**

The choice to use the Stream API for sorting is driven by the need for a concise and modern approach to sorting the HashMap entries. This eliminates the need for manually iterating and sorting, making the code cleaner and more readable. It also allows for easy customization, should future requirements dictate a different sorting mechanism.

**c. File Operations**

The program's handling of file operations, such as reading from and writing to files, is straightforward. Error handling is present to ensure that if an IOException occurs, the user is informed and the program doesn’t crash unexpectedly.

**d. User Interaction**

The program prompts the user for input in a clear and structured manner. The use of a switch statement for handling gender-based input adds simplicity to the decision-making process. The search functionality is intuitive, allowing the user to search for a name and view its rank based on the data.