**Project #2 Word Statistics Documentation**

* **Overview**

The Word Statistics project is designed to analyze text files within a specified directory and its subdirectories, providing real-time updates in a graphical user interface (GUI). The main thread is responsible for identifying text files, displaying them in the GUI, and coordinating the exploration of these files by multiple threads. Each exploration thread processes one or more text files, and the number of threads is determined by the available processors (cores). The GUI allows users to input the directory path, choose whether to include subdirectories, and presents output in a table format, including word counts, occurrences of specific words, and information about word lengths.

* **GUI Features**

**Input**

1. **Browse Button:** Allows users to select the directory using a file dialog.
2. **Checkbox for Including Subdirectories:** Enables users to specify whether to include files in subdirectories during the analysis.

**Output**

**The GUI displays real-time updates in table form with the following information:**

* **Word Counts:** The number of occurrences for each of the specified words (#words, #is, #are, #you).
* **Longest Word per File:** The longest word found in each processed file.
* **Shortest Word per File:** The shortest word found in each processed file.
* **Longest Word per Directories:** The longest word found in each explored directory.
* **Shortest Word per Directory:** The shortest word found in each explored directory.
* **Technical Details**

**Multithreading**

The main thread is responsible for identifying text files and managing the GUI.

Multiple worker threads are created based on the number of available processors (cores).

Each worker thread explores one or more text files concurrently.

**Communication**

Worker threads send real-time updates to the GUI to ensure the display is continuously updated.

Updates include word counts, longest and shortest words per file, and per directory.

* **Code Overview**

**ReadFiles Class**

The ReadFiles class facilitates the retrieval of directory and file paths within a specified file system. It maintains two private lists, pathsDirs for directories and pathsFiles for files. The class offers two main methods:

**getDirectories Method:**

* Purpose:

Recursively obtains directory paths up to a specified depth level.

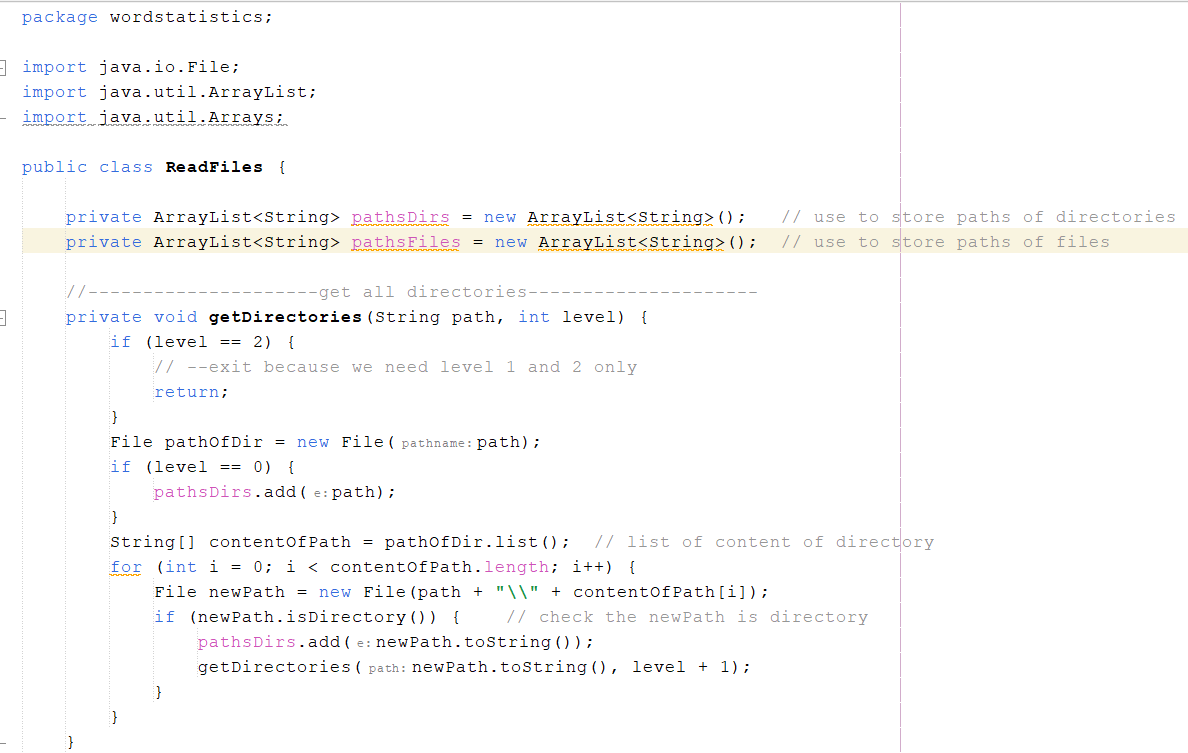
* Parameters:
  + path (String): Starting directory path.
  + level (int): Depth level (up to level 2).
* Behavior:
  + Directories are added to pathsDirs.
  + Stops recursion at level 2.

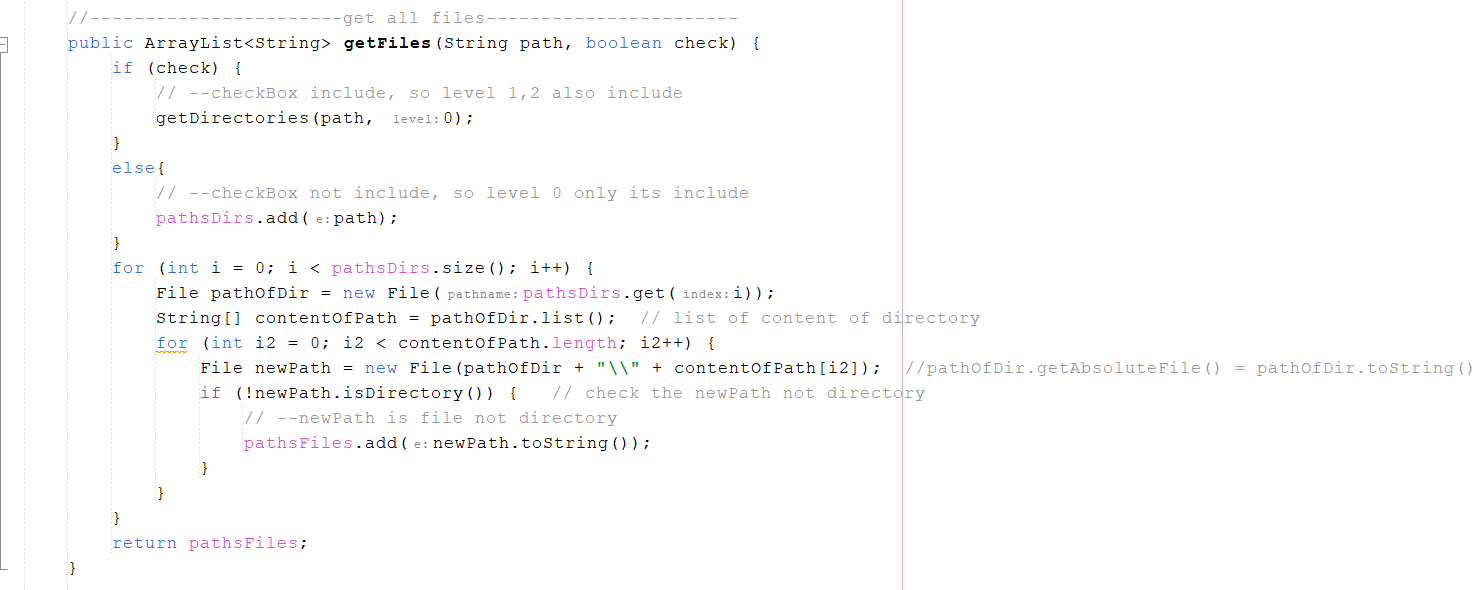
**getFiles Method:**

* Purpose:

Retrieves file paths within a specified directory, with an option to include subdirectories.

* Parameters:
  + path (String): Starting directory path.
  + check (boolean): If true, includes subdirectories (levels 1 and 2).
* Behavior:
  + Utilizes getDirectories if subdirectories are to be included.
  + Adds file paths to pathsFiles.



****

**ReadFiles Class**

The **BuildThread** class orchestrates the parallel processing of file operations using multithreading. Key functionalities include:

* **Constructor:**
  + The class has a method named **BuildThread**, which serves as a constructor. However, the naming convention suggests it may be intended as an initializer method.
* **Multithreading File Processing:**
  + Utilizes the **ReadFiles** class to obtain paths of files within a specified directory.
  + Creates an instance of **RunnableClass** (assumed to be a class implementing the **Runnable** interface) and sets the obtained file paths using the **setfiles** method.
  + Initiates four threads (**t1**, **t2**, **t3**, **t4**) to concurrently execute the **RunnableClass** logic.
* **Thread Creation and Start:**
  + Creates four threads from the same instance of **RunnableClass**.
  + Each thread independently processes the file paths using the logic defined in **RunnableClass**.
  + Threads (**t1**, **t2**, **t3**, **t4**) are started simultaneously.
* **Important Notes:**
  + The **ReadFiles** class is assumed to be properly implemented and functional.
  + The behavior of the threads depends on the logic defined within the **RunnableClass**.
  + Ensure appropriate error handling for potential file system access issues or invalid paths.
  + It's recommended to adhere to Java naming conventions (methods typically start with a lowercase letter).

A screenshot of a computer

Description automatically generated

**RunnableClass Class**

The **RunnableClass** class implements the **Runnable** interface to define a threadable unit of work. It processes text files in a multithreaded manner, extracting word statistics and updating a graphical user interface (GUI) with the results.

**Class Fields:**

* **filesPath** (**ArrayList<String>**): A list holding the paths of files to be processed.
* **sleepTime** (**int**): A constant representing the sleep time in milliseconds between GUI updates.

**Methods**:

1. **setfiles Method:**
   * **Purpose:**
     + Sets the file paths to be processed by the thread.
   * **Parameters:**
     + **S** (**ArrayList<String>**): List of file paths.
   * **Return Type:**
     + **void**
2. **geFilesSize Method:**
   * **Purpose:**
     + Retrieves the current size of the list of file paths.
   * **Return Type:**
     + **int**
3. **getFilePath Method:**
   * **Purpose:**
     + Retrieves and removes the first file path from the list in a synchronized manner.
   * **Return Type:**
     + **String**
4. **run Method (Implemented from Runnable):**
   * **Purpose:**
     + Defines the main logic executed when a thread is started.
   * **Behavior:**
     + Reads each line of a file, counting occurrences of specific words, finding the longest and shortest words, and updating the GUI with statistics.
     + Utilizes synchronization for thread safety.
     + Introduces a delay (**sleepTime**) between GUI updates.
   * **Return Type:**
     + **void**

**Important Notes:**

* The class assumes a GUI class named **GuiAndUpdate** with a static method **showUpdateData** for updating the GUI with statistics.
* Error handling for file not found and interrupted thread exceptions is included.
* Word statistics include counts of "is," "are," and "you," as well as the longest and shortest words in each line.
* The class dynamically extracts file names and statistics, updating the GUI for each line in each file.
* Ensure proper initialization of the **filesPath** list before starting the threads.

A screenshot of a computer program

Description automatically generated



A screenshot of a computer code

Description automatically generated

**ReadFiles Class**

he **showUpdateData** method is a synchronized static method responsible for updating a graphical user interface (GUI), specifically a table and additional UI elements. The method takes a list of statistics for a file line (**receiveNewLine**) and dynamically updates the GUI based on the received data.

**Parameters:**

* **receiveNewLine** (**ArrayList<String>**): List containing statistics for a file line, including file name, word counts, longest and shortest words.

**Method Logic:**

1. **Search for Existing Row:**
   * Iterates through the rows of a table (**storeTableModel**) to find an existing row that corresponds to the received file name.
   * If an existing row is found, the loop breaks.
2. **Update Existing Row:**
   * If an existing row is found (**row != storeTableModel.getRowCount()**), updates the values in the existing row with the received statistics.
   * Calculates the sum of word counts and updates the corresponding columns.
3. **Add New Row:**
   * If no existing row is found (**row == storeTableModel.getRowCount()**), adds a new row to the table with the received statistics.
4. **Notify Table Data Change:**
   * Notifies all registered listeners that the table data has changed.
5. **Update Overall Longest and Shortest Words:**
   * Compares the received longest and shortest words with the current overall longest and shortest words displayed in the GUI.
   * Updates the GUI elements (**acuallylongestWord** and **acuallyShortestWord**) with the latest longest and shortest words.

