# Abbreviation Class

## Program Logic & Justifications

### Constructor Detail

***public Abbreviation (String abbrName, String fullName)***

Initializes a newly created Abbreviation object using the argument strings.

***public Abbreviation (Abbreviation abbr)***

Initializes a newly created Abbreviation object using data from the argument Abbreviation. Loops through the argument object’s fullName array in case there is more than one full name.

### Method Detail

***public String getAbbrName()***

Returns a copy of the abbreviation’s name.

***public int getNumFullNames()***

Returns the number of full names of the abbreviation.

**public String getFullName(int index)**

Returns a copy of the abbreviation’s full name located at the specified index of the fullNames array.

***public void addFullName(String fullName)***

Adds a new, single full name to the abbreviation. The fullNames array is resized if it overflows.

***public boolean matchByAbbrName(String abbrName)***

Returns true if, and only if, the argument is the same as the abbreviation’s name.

***public boolean matchByAbbrPrefix(String abbrPrefix)***

Returns true if, and only if, the argument is a prefix of the abbreviation’s name; in other words, the argument string is a substring of the abbreviation’s name starting at index 0.

***public void print()***

Prints all full names of the abbreviation onto the console.

## Suggestions

## None

# Database Class

## Program Logic & Justifications

### Constructor Detail

***public Database (String filename)***

Initializes a newly created Database object with a list of abbreviations. The abbreviations are obtained from the filename supplied to the argument. Initialized Abbreviation objects are added to an array list until all abbreviations are obtained. The array list is then type casted to an Abbreviation array and stored in the class attribute.

**Method Detail**

***private void add(ArrayList<Abbreviation> al, String abbr, StringBuffer abbrName)***

Adds a newly initialized Abbreviation object to an array list during database initialization.

***private int exists(ArrayList<Abbreviation> al, String abbr)***

Searches for an existing abbreviation and returns its index during database initialization. Search is done sequentially as database elements are not yet sorted at that time.

***public int getNumAbbreviations()***

Returns the total number of abbreviations.

***public Abbreviation getAbbreviationByIndex(int index)***

Returns a copy of the abbreviation object located at the specified index.

***public void printAllAbbreviations()***

Print all abbreviation names and their full names to console.

***private void sort(Abbreviation [ ] a, int start, int end)***

Recursively sorts all elements in the class attribute after initialization using quicksort algorithm. Calls the quickSort() method for the actual sorting. Prerequisite for the BinarySearch() method.

***private int quickSort(Abbreviation [ ] a, int start, int end)***

Sort all elements in the specified range of the argument array with respect to the middle element. Returns the final position of the original leftmost pointer after the sort.

***public Abbreviation queryByAbbrName(String target)***

Searches for and returns a copy of the abbreviation whose name matches the argument string to prevent external modification. Calls the BinarySearch() method for the actual searching.

**private Abbreviation BinarySearch(Abbreviation [ ] abbr, String target, int start, int end)**

Recursively searches for the abbreviation whose name matches the argument string and returns its copy. Search is case-insensitive.

**public Abbreviation [ ] queryByAbbrPrefix(String abbrPrefix)**

Searches for all abbreviations that have the string argument prefix and returns a new sub-array. Search is done sequentially as there can be more than one result.

## Suggestions

1. Insertion-sort can be implemented to ensure the class attribute is always sorted. This will allow for binary searching of existing abbreviations even during database initialization.
2. A simple indexing system can be implemented to indicate the starting element of each alphabet within the class attribute. This narrows down the range of values searched for any given prefix.

# TranslateApp Class

## Program Logic & Justifications

### Constructor Detail

***public TranslateApp (Database database)***

Initializes a newly created TranslateApp object by pointing its class attribute to the memory location of an existing Database object.

**Method Detail**

**public int translateFile(String input\_filename, String output\_filename, int max\_linewidth)**

Reads in a file, translates any abbreviations and appends both the transcript and translation to a temporary string buffer which is written to an output file. Returns the total number of translated abbreviations.

**public boolean isAbbreviation(String token)**

Returns true if, and only if, the string argument is deemed to be an abbreviation; in other words, it is in all uppercase.

**private int digDeeper(StringBuffer outputHolder, String token, int translated)**

Reads the stored string token immediately after an apostrophe token, otherwise it will be lost. Returns the new number of abbreviations translated, if any.

**private void appendAbbreviations(StringBuffer outputHolder, Abbreviation a)**

Appends the translation of an abbreviation to the string buffer argument.

**private void writeToFile(StringBuffer output, int limit, String output\_filename)**

Writes the string buffer to an output file, with respect to the number of characters per line allowed.Code base is heavily recycled from the translateFile() method.

## Suggestions

None

# MainApp Class

## Program Logic & Justifications

**Method Detail**

***private static void main(String [ ] args)***

Main entry point for the entire program. Prompts user for desired operation and provides transparent interfacing with other classes.

***private static boolean checkPointOne()***

First round of checks before proceeding with file translation. Returns true when, and only when, sure that the input file exists and user has read privileges.

**private static boolean checkPointTwo()**

Second round of checks before proceeding with file translation. Returns true when, and only when, user provides a valid input giving explicit permission to overwrite an existing output file.

**private static boolean checkPointThree()**

Final round of checks before proceeding with file translation. Returns true when, and only when, user provides a valid input for the maximum line width of the output text.

## Suggestions

None