פרויקט בניית מנוע לאחזור מסמכים

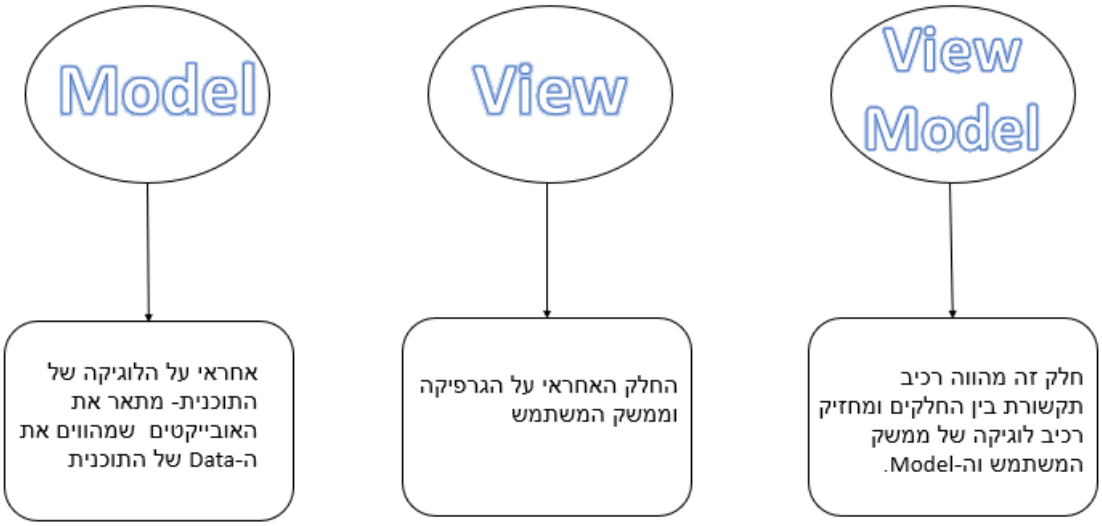
אביחי צרפתי 204520803

ירדן לוי 204341580

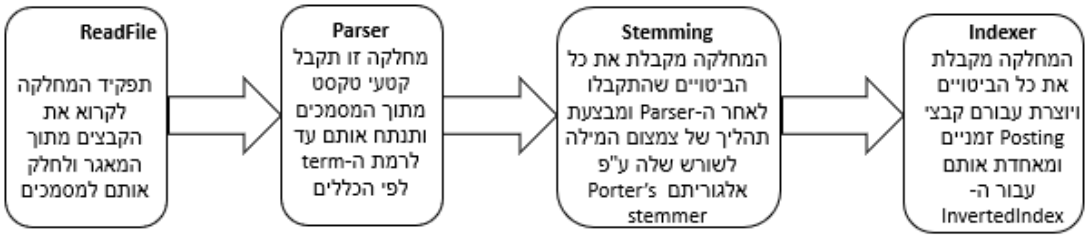
372-1-4406, 2019-2020, סמסטר חורף
קורס אחזור מידע, המחלקה להנדסת מערכות תוכנה ומידע

חלק א'- עיצוב התוכנה:

בבניית התוכנית בחרנו להשתמש בתבנית הארכיטקטורה MVVM על מנת ליצור סדר בין כל האובייקטים השונים והמשימות השונות שנדרש לעמוד בהן במסגרת פרויקט המנוע. כמו כן, באמצעות תבנית זו ניתן לשמור בצורה טובה על עקרונות ה-SOLID כפי שלמדנו.



מנוע אחזור המסמכים מחולק למחלקות ושיטות רבות. התהליך העיקרי בפרויקט הינו:



כעת נפרט את כלל המחלקות והפונקציות בפרויקט בשכבת ה-Model:

מחלקת ReadFile:

מחלקה זו קוראת את כל הקבצים ממאגר ה-corpus ומחלקת כל קובץ לפי המסמכים השונים בו.

```
public LinkedList<cDocument> readFiles (String pathOfDocs, int mone, int mechane)
```

שיטה זו אחראית לקרוא את כל הקבצים במאגר ואת המסמכים שמוכלים בהם. השיטה תקבל את הנתיב למאגר המסמכים ושני משתנים נוספים לקביעת כמות ואופן יצירת קבצי ה-Posting הזמניים.

השיטה תבצע קריאה במקביל של כל הקבצים ותפריד אותם לפי מסמכים (על פי תגית <DOC>). כלומר, עבור כל ליבה במחשב יפעלו שני תהליכים במקביל וכך תיווצר קריאה מהירה ויעילה של כלל המסמכים.

בסוף השיטה תוחזר רשימה מקושרת כאשר כל איבר בה הינו אובייקט מסוג מסמך.

```
public static HashSet<String> initSet (String fileName)
```

TODO

```
class Reader implements Callable<LinkedList<cDocument>>
```

class בתוך מחלקת ReadFile כך שתממש את הממשק Callable המאפשר החזרת ערך עם סיום ריצת Threads ובנוסף מאפשר זריקת שגיאות מתאימות בעת הצורך.

<u>נפרט את השיטות בתוך class Reader:</u>

```
public LinkedList<cDocument> call()
```

שיטה מחייבת לצורך מימוש הממשק Callable ופעולת התוכנית ע"י תהליכים.

```
private LinkedList<cDocument> read (File fileToSeparate)
```

השיטה מקבלת קובץ מתוך מאגר הקבצים ובאמצעות קוד פתוח Jsoup היא מפרידה בין כל המסמכים השונים ע"י התגית <DOC> ויוצרת אובייקטים מסוג cDocument. Jsoup הוא קוד אשר עובד עם שפת HTML ומחלק כל קובץ לפי תגיות מוגדרות. אנחנו מחלקים כל קובץ למסמכים על מנת שנוכל לשלוח אותם ל-Parser ולהעביר אותם את כל תהליך האחזור לפי הכללים. בנוסף, אנחנו מפרידים את קטעי הטקסט (<TEXT>) אותם אנחנו מאנדקסים.

השיטה מחזירה רשימה מקושרת המכילה את האובייקטים מסוג cDocument.

```
public class cDocument
```

class אשר מייצגת אובייקט של מסמך יחיד מתוך קובץ מסוים. השדות שתחזיר הינם שם הקובץ אליו שייך המסמך, וקטע הטקסט הקיים במסמך.

class cDocument מחזיקה שתי שיטות get בלבד להחזרת הערכים של השדות שלה.

<u>מחלקת WriteFile:</u>

המחלקה הינה מחלקה מקשרת אשר אחראית לכתוב קבצי posting זמניים, ליצור מילון לכל מסמך עם/בלי stemming (בהתאם להחלטת המשתמש, ליצור את הקבצים ההופכיים ולייצר קבצי posting סופיים לפי הכללים הנדרשים. פעולת מחלקה זו מתרחשת לאחר ביצוע תהליך ה-Parse.

```
public static void writeTempPosting (String path, int postingNum , HashMap<String,
Pair<Integer,StringBuilder>> temporaryPosting)
```

השיטה אחראית ליצור קובץ Posting זמני ולשם כך מקבלת את כל המידע הנדרש: הנתיב לתיקייה במחשב אליה ייכתב הקובץ הזמני, מספר הקובץ וטבלת Hash עם תוכן הקובץ.

```
public static void writeDocDictionary(String path, HashMap<String, DocDictionaryNode>
documentDictionary, boolean stem)
```

השיטה אחראית לכתוב מילון עבור מסמך מסוים. השיטה מקבלת נתיב לתיקייה אליה ייכתב המילון, בנוסף לכך היא תקבל גם טבלת Hash אשר כוללת את כל המידע על המסמך כאשר ה-value בטבלה הינה אובייקט מסוג DocDictionaryNode. כמו כן, השיטה תקבל ערך בוליאני אשר יגדיר האם יש לבצע Stemming.

```
public static void writeInvertedFile(String path, InvertedIndex invertedIndex, boolean
stem)
```

השיטה מבצעת כתיבה בפועל של הקובץ ההופכי שיצרנו במחלקת InvertedIndex. השיטה תקבל נתיב לתיקייה אליה ייכתב הקובץ ההופכי שגם הוא מתקבל כמשתנה, ובנוסף נקבל משתנה בוליאני אשר יגדיר אם יש לבצע Stemming.

```
public static void writeFinalPosting(String fileName, HashMap<String, StringBuilder>
finalPosting)
```

השיטה אשר אחראית לכתוב את קובץ ה-Posting הסופי. בתהליך הכתיבה של הקובץ הסופי נבצע מיזוג לכל קבצי ה-Posting הזמניים וניצור קבצים חדשים כך שיחולקו לפי אותיות ה-ABC (כלומר, לאות a יהיה קובץ Posting סופי משלה וכך הלאה), בנוסף לקובץ עבור מספרים. בתהליך המיזוג עד ליצירת הקבצים הסופיים נשמור על סדר עולה לפי ה-ABC ו/או מספרים וסימונים שונים. כל שורה בקובץ הסופי תציג את ה-term וכל המידע הדרוש: עבור כל מסמך בו ה-term מופיע נקבל את שם המסמך, כמות הפעמים אשר ה-term מופיע בו (tf), מיקום ה-term במסמך זה. בסוף השורה תופיע כמות הופעות ה-term במאגר כולו (df).

כך עבור כל term.

```
private static void write(File actualFile, StringBuilder toWrite)
```

שיטה שמבצעת פעולת כתיבה לקובץ עבור ערך מחרוזת מסוים.

<u>מחלקת InvertedIndex:</u>

מחלקה זו אחראית על יצירת קובץ הופכי עבור מסמך.

```
public InvertedIndex(File file)
```

השיטה העיקרית במחלקה אשר יוצרת קובץ הופכי חדש- השיטה תקבל קובץ אשר עבורו יש ליצור את הקובץ ההופכי המתאים.

בשיטה זו נעבור על הקובץ שהתקבל, שורה אחר שורה, ועבור כל שורה נפריד את המידע הנדרש עבור ה-term:

1. שם ה-term.
2. מספר המסמכים השונים בהם הוא מופיע.
3. שכיחות הופעת ה-term במאגר.
4. מספר השורה בה הוא מופיע בקובץ ה-posting.

```
public HashMap<String, Pair<Integer, StringBuilder>> call()
```

TODO

```
public void addTerm (String term)
```

השיטה אחראית להוסיף term לקובץ ההופכי, או במידה והוא כבר קיים- היא תעדכן את ה-tf עבורו.

```
public void deleteEntriesOfIrrelevant()
```

השיטה אחראית למחוק רשומות עבור term שאינם רלוונטיים עבור הישויות שאנו נדרשים לשמור.

```
public int getNumOfUniqueTerms()
```

השיטה מחזירה את כמות ה-term הייחודים.

```
public void setPointer(String minTerm, int lineNumber)
```

השיטה בודקת האם ה-term שהתקבל במשתנה minTerm אכן קיים בקובץ ההופכי ובמידה וכן מעדכנת עבורו מצביע לשורה בקובץ ה-Posting.

```
public void setNumOfAppearance(String term, int numOfAppearance)
```

השיטה בודקת האם ה-term שהתקבל במשתנה minTerm אכן קיים בקובץ ההופכי ובמידה וכן מעדכנת עבורו את מספר הופעותיו.

```
public ObservableList<ShowDictionaryRecord> getRecords ()
```

שיטה אשר מחזירה רשימה המכילה את כל רשומות הקובץ ההופכי.

```
class Term
```

זו class בתוך מחלקת InvertedIndex והיא תייצג עבורו term יחיד. שדות ה-class הינם:

1. שם ה-term.
2. מספר המסמכים השונים בהם הוא מופיע.
3. שכיחות הופעת ה-term במאגר.
4. מספר השורה בה הוא מופיע בקובץ ה-posting.

המידע שיישמר עבור כל term ישמש אותנו ליצירת הקובץ ההופכי.

בתוך class Term קיימות שיטות מסוג get ו-set בלבד אשר אחראית לעדכון והחזרת ערכי שדות ה-class בהתאם לצורך. כמו כן קיימת שיטת toString שתפקידה להחזיר מחרוזת עם שדות ה-class.

```
public class ShowDictionaryRecord
```

תת מחלקה שמטרתה לאפשר צפייה בנתוני המילון ולבצע מעטפת לנתוני ה-term.

<u>מחלקת MiniDictionary:</u>

מחלקה שמייצגת מילון עבור כל מסמך (DOC). שדות המחלקה הם:

1. מבנה HashMap שמייצג מילון שבו כל key הינו term וכל value הינו ה-tf שלו במסמך.
2. שם המסמך עבורו נוצא ה-miniDictionary.
3. מספר המופעים של ה-term השכיח ביותר במסמך.

4. ה-term השכיח ביותר במסמך.
5. כותרת המסמך.
6. מערך המייצג את 5 המילים הייחודיות ביותר במסמך.

```
public void addWord (String word, int placeInText)
```

השיטה מקבלת term ומיקומו במסמך ומוסיפה אותו למילון. במידה וקיים כבר אז נעדכן את כמות ההופעות שלו במסמך ובמידת הצורך נעדכן גם את המילה השכיחה ביותר.

```
private int containsKey (String word)
```

השיטה בודקת האם ה-term כבר קיים במילון בצורה כל שהיא- אותיות קטנות בלבד, אותיות גדולות בלבד, אות ראשונה גדולה וכדומה. בהתאם לכך נחזיר תשובה ונשמור את ה-term בצורה המתאימה לפי הכללים.

```
public String listOfData (String word)
```

השיטה תקבל מחרוזת המייצגת term ותחזיר את כל המידע הידוע עליו: ה-term עצמו, שכיחות במסמך וכל המיקומים שלו במסמך.

```
private String printIndexes (LinkedList<Integer> indexesOfWord)
```

שיטה המדפיסה את כל ה-Positions של term מסוים במסמך. השיטה תקבל רשימה של Positions שכבר שייכים ל-term ספציפי ותבצע הדפסה שלהם לפי התצורה הסופית בה יופיעו במילון.

```
public int size()
```

השיטה מחזירה את גודל המילון, כלומר, כמות ה-term הנמצאים בו.

```
private LinkedList<Integer> getIndexesOfWord (String word)
```

השיטה תקבל term ותחזיר רשימה המכילה את כל ה-Positions שלו במסמך בו אנו נמצאים.

```
public String getMaxFreqWord ()
```

השיטה תחזיר את שם ה-term השכיח ביותר במסמך בו אנו נמצאים.

```
public String getName ()
```

השיטה תחזיר את שם המסמך בו אנו נמצאים.

```
public int getFrequency (String word)
```

השיטה תקבל מחרוזת המייצגת term ותחזיר את השכיחות שלו במסמך בו אנו נמצאים.

```
public int getMaxFrequency ()
```

השיטה תחזיר את השכיחות המקסימלית במסמך בו אנו נמצאים- כלומר, את כמות הפעמים שמופיע במסמך ה-term השכיח ביותר.

```
public int getDocLength ()
```

השיטה תחזיר את אורך המסמך בו אנו נמצאים.

```
public Pair<String, Integer>[] getPrimaryWords()
```

השיטה תחזיר מערך מסוג KeyValuePair המכיל את 5 ה-terms השכיחים ביותר, ואת שכיחותם במסמך בו אנו נמצאים.

```
public String getTitle ()
```

השיטה תחזיר מחרוזת המייצגת את כותרת המסמך בו אנו נמצאים.

מחלקת DocumentIndex:

מחלקה אשר מייצגת מסמך ותשמש אותנו בניהול תהליך הפרסור וכתיבת קבצי ה-Posting. המחלקה תחזיק את שם המסמך, כמות ה-terms הייחודיים במסמך, ה-term הייחודי ביותר במסמך ושכיחותו, אורך המסמך, כותרת המסמך.

מחלקת cDocument:

מחלקה המייצגת מידע על מסמך ישירות לאחר שנקרא מקובץ מסוים. המחלקה תחזיק את שם הקובץ ממנו המסמך נקרא, את מספר המסמך, תאריך המסמך, כותרת המסמך והטקסט שנמצא בו (לפי תגית TEXT).

מחלקת Parse:

מחלקה זו לוקחת טקסט של מסמך ומעבירה אותו תהליך ניתוח לרמת ה-terms. בתהליך זה המחלקה תדע לסווג כל term למשמעות והמהות המתאימה לו על פי הכללים הנדרשים.

```
public MiniDictionary call ()
```

השיטה זו מפעילה את שיטת parse אשר מבצעת את כל תהליך ניתוח הטקסט עבור מסמך מסוים ומחזירה מיני-מילון עבור המסמך או שגיאה במקרה של תקלה בתהליך.

```
public MiniDictionary parse ()
```

שיטה זו אחראית על כל תהליך פרסור הטקסט. השיטה תקרא עבור כל מסמך את הטקסט שיוחזר ממחלקת ReadFile. עבור כל קטע טקסט ששייך למסמך (DOC) השיטה תבצע ניתוח של הטקסט, מיון לביטויים ושמירה במשתנה Term. השיטה תייצר מיני-מילון שהוא למעשה מילון עבור מסמך ספציפי אותו העברנו את התהליך.

```
private String numberValue (Double d)
```

השיטה מחזירה את הערך ב-int של המשתנה שהתקבל מסוג double.

```
private boolean isInteger (double word)
```

השיטה בודקת האם המספר הוא Integer.

```
private String handleDollar (String price, boolean containsComma)
```

השיטה מקבלת מחרוזת שמייצגת מספר שהוא מחיר כל שהוא בדולרים ומשתנה בוליאני אשר אומר האם המספר גדול מספיק כדי להכיל ",". כלומר, האם המספר הינו גדול או שווה ל-1,000. השיטה תחזיר מחרוזת של המחיר על פי כללי ה-Parser.

למשל עבור המחרוזת "1,000,000" תוחזר המחרוזת "1 M Dollars".

```
private String addCommas (String number)
```

השיטה תקבל מחרוזת שמייצגת מספר ותוסיף , או . בהתאם לדרישה בכללי ה-Parser.

```
private LinkedList<String> stringToList (String [] split)
```

השיטה תקבל מערך שמורכב ממחרוזות ותחזיר אותו כרשימה.

```
private boolean isRangeNumbers (String s)
```

השיטה בודקת האם המחרוזת כוללת שני מספרים שהם חלק מטווח (למשל "18-24") ומחזירה true או false בהתאם.

```
private String handleWeight (String term, String unit)
```

TODO

```
private boolean checkIfFracture (String token)
```

השיטה בודקת האם המספר שהתקבל כמחרוזת הוא למעשה חלק משבר. למשל עבור "3/4" השיטה תחזיר true.

```
private String cleanTerm (String term)
```

השיטה מקבלת מחרוזת עם ביטוי מסוים ומחזירה את הביטוי ללא סימנים נוספים כגון "&", "%", "$" וכדומה.

```
private String handleNumber (String number)
```

השיטה מטפלת במחרוזות שמייצגות מספרים רגילים. השיטה תקבל מחרוזת המייצגת מספר ומחזירה מחרוזת תקנית לפי כללי הפרסור. למשל עבור "1,973.45" תוחזר המחרוזת "1.973 K".

```
private String nextWord ()
```

שיטה זו מחזיקה את הרשימה המקושרת wordlist אשר כוללת את כל המילים בטקסט של המסמך עליו אנו עובדים, החל מהמילה הבאה שיש לנתח.

```
private double threeDigit (String check)
```

שיטה זו מטפלת במספרים רגילים. השיטה תקבל מחרוזת המייצגת מספר ומוודאת כי לאחר הטיפול בו (במקרה של B/M/K או במקרה שהמספר הינו עשרוני) נשארות רק 3 ספרות אחרי הנקודה.

שיטות nextWordRules() ו-initMonthData() מייצגות מבני נתונים מסוג HashMap לצורך שליפה מהירה של ערכי term על פי כללי הפרסור הנדרשים. למשל, כל term המייצג שם של חודש יוחזר לפי המספר המתאים. כגון APRIL -> 04.

מחלקת Stemmer:

במחלקה זו השתמשנו בקוד פתוח המייצג את Porter's stemming algorithm. מהות מחלקת Stemmer היא לצמצם מילים שאנו מוצאים בתהליך קריאת המסמך והפרסור שלו ע"י הורדת תוספות סופיות מיותרות למילה. לצורך כך נעשה שימוש בקוד זה אשר באמצעות האלגוריתם מזהה consonant ו-vowels בכל מילה ומצמצם את הנדרש לפי כללי האלגוריתם.

להלן קישור להסבר על פעולת האלגוריתם והחוקים כפי שנלמד בהרצאה ובתרגול:

http://snowball.tartarus.org/algorithms/porter/stemmer.html

מחלקת NamedEntitiesSearcher:

במחלקה זו נעשה שימוש בקוד פתוח StanfordCoreNLP עבור רמה 3 שהינה הרמה הבסיסית ביותר בקוד. קוד זה תורם בתחום ניתוח ישויות ובהינתן קטעי טקסט מורכבים ידע להפריד כל term ולסווג אותו על פי צורך. למשל, בהינתן המשפט John Smith is in Manhattan New-York האלגוריתם ידע לסווג את " John Smith" כשם של בן אדם (PERSON), את Manhattan כשם של עיר (CITY) ואת New-York כמדינה בהקשר זה (COUNTRY).

לאלגוריתם ישנן מספר שכבות של עומק בניתוח טקסט אשר להן אנו לא נדרשים בפרויקט זה. לכן, בחרנו להשתמש בשכבה הבסיסית ביותר ולבצע הכנה שתאפשר לנו שימוש בקוד במידת הצורך.

השיטה היחידה במחלקה buildPipeLine מגדירה את ה-Properties של הקוד. כלומר, בשיטה זו נגדיר אילו אספקטים נרצה שהאלגוריתם ינתח ואילו לא.

מחלקת Model:

כפי שהגדרנו בתחילת ההסבר על עיצוב התוכנה, שכבת ה-Model אחראית על הלוגיקה של התוכנית ומחלקת Model הינה החלק המרכזי של שכבה זו, אחראית לניהול כלל התהליכים הפועלים בתוכנית והשלמתם בצורה מיטבית.

```
public void startIndexing(String pathOfDocs, String destinationPath, boolean stm)
```

שיטה זו אחראית על תהליך האינדוקס- מקבלת את הנתיב בו נמצאים כל קבצי המאגר, נתיב לתיקייה אליה ייכתבו כל הקבצים הסופיים, ומשתנה בוליאני אשר מגדיר האם יש להשתמש בתהליך ב-stemming או לא.

השיטה תבדוק האם הנתיבים שהוכנסו על ידי המשתמש הינם חוקיים. אם אכן חוקיים השיטה תפעיל את ביצוע התוכנית ע"י השיטה המרכזית במחלקה זו – mainLogicUnit.

במקרה של תקלה בריצת התוכנית השיטה תוציא הודעה מתאימה. במידה והתוכנית סיימה לרוץ בהצלחה תצא הודעה מתאימה עם סיכום הנתונים- כמות terms שנותחו, הזמן הכולל שלקח לתוכנית לרוץ מתחילתה ועד סופה ומספר המסמכים שנקראו.

```
private int[] mainLogicUnit(InvertedIndex invertedIndex, String corpusPath, String
destinationPath, boolean stem) throws Exception
```

השיטה העיקרית במחלקה ובפרויקט אשר מנהלת את כל התהליך מתחילתו ועד סופו. השיטה תקבל קובץ הופכי, נתיב לתיקייה ממנה נקרא את מאגר המסמכים, נתיב לתיקיית היעד ומשתנה בוליאני שיגדיר האם יש לבצע stemming או לא.

במהלך ריצת התוכנית נדפיס בlogger הודעות על התקדמות התוכנית: Start Parsing and Indexing, Start runnable :: manager Method וכדומה.

השיטה תיצור רשימת אובייקטים מסוג cDocuments אשר אליה ייכנסו כל המסמכים שייקראו משיטת readFiles של מחלקת ReadFile. בנוסף, ייווצר thread pool שיווסת את המסמכים השונים לשם יצירת המקביליות. השיטה תיצור אוסף של miniDictionary שייווצר לאחר תהליך הפרסור ולבסוף השיטה תבצע את תהליך האינדוקס על ידי קבצי Posting זמניים. לבסוף נבצע מיזוג לכדי קבצי Posting סופיים וקובץ הופכי יחיד.

```
public void loadDictionary (String path, boolean stem)
```

השיטה תבצע טעינה של המילון מהנתיב שהתקבל לשם כך או תוציא הודעה מתאימה במידה ונתקלה בשגיאה.

```
private String [] pathsAreValid(String pathOfDocs, String destinationPath)
```

השיטה תקבל את הנתיב לתיקיית מאגר הקבצים והנתיב לתיקיית היעד אליה ייכתבו הקבצים הסופיים ותבדוק האם הקבצים שהתקבלו הינם תקניים. במידה ולא, השיטה תוציא הודעה מתאימה. במידה וכן, השיטה תחזירה מערך שיכלול מחרוזות המייצגות את הנתיב לתיקיית מאגר המסמכים, נתיב לתיקיית היעד, ונתיב לקובץ stop_words.

```
private void insertData (ConcurrentLinkedDeque<MiniDictionary> miniDicList, InvertedIndex
invertedIndex)
```

השיטה אחראית להכניס את כל המידע הנדרש לתוך רשימה של miniDictionary.

```
private void mergePostings (InvertedIndex invertedIndex, String tempPostingPath, boolean
stem) throws IOException
```

השיטה אחראית על מיזוג כל קבצי ה-Posting הזמניים שנשמרו במהלך הרצת התוכנית. בתהליך זה ייווצרו הקבצים הסופיים אשר יחולקו באופן הבא:

1. קובץ טקסט הופכי אשר כל שורה בו מייצגת term ואת המידע הנדרש לגביה כפי שפורט במחלקת invertedIndex.
2. קובץ טקסט מילון אשר כל שורה בו מייצגת קובץ ואת המידע הנדרש לשמור לגביו.
3. קובץ עבור כל אות ב-ABC אשר כל שורה בו מייצגת term כאשר עבור כל term נראה בקובץ את שמות המסמכים בהם הוא מופיע, כמות הפעמים שהוא מופיע בכל מסמך כזה, ואת המיקומים שלו בכל אחד מהמסמכים האלו.

```
private void lookForSameTerm (String minTerm, StringBuilder finalPostingLine,
HashMap<String, StringBuilder> writeToPosting)
```

השיטה אחראית לאתר כפילויות בין terms. במידה ו-term כבר נשמר בתצורה כל שהיא (אותיות גדולות בלבד/ אותיות קטנות בלבד וכדומה) השיטה תאתר זאת ותמנע אפשרות של שמירה כפולה של אותו ה-term. בנוסף, השיטה תעדכן את המידע הנדרש.

```
private StringBuilder separator (String minTerm, StringBuilder finalPostingLine, String
replace)
```

השיטה אחראית על עיצב הנראות של קובץ ה-Posting הסופי. למשל, כל שורה בקובץ תהיה מהצורה:

כלומר (משמאל לימין)- שם ה-term, שם מסמך בו הוא מופיע, כמות הפעמים בו הוא מופיע במסמך, ה-position שלו באותו מסמך ולבסוף כמות הפעמים שיופיע במאגר כולו.

```
private void restoreSentence (LinkedList<BufferedReader> bufferedReaderList, String
minTerm, String[] firstSentenceOfFile, String[] saveSentences)
```

השיטה, יחד עם השיטה הקודמת separator, אחראית על שמירת כל שורה בקובץ הסופי על פי הסדר שהוגדר ובהתאם לכללים.

```
private String getNextSentence(BufferedReader bf)
```

השיטה מחזירה את השורה הבאה מהקובץ.

```
private String[] initiateMergingArray(LinkedList<BufferedReader> bufferedReaderList)
```

השיטה מקבלת את הרשימה שהתקבלה מהשיטה initiateBufferedReaderList ולוקחת כל איבר בה כמחרוזת לתוך מערך אותו השיטה תחזיר. זאת למעשה הכנה לפעולת ה-merge שנבצע.

```
private LinkedList<BufferedReader> initiateBufferedReaderList (String tempPostingPath)
```

השיטה אחראית על יצירה ואתחול ה-bufferedReaderList והחזרתה.

```
public void startOver (String path)
```

שיטה שנועדה לשם פעולת ה-reset. השיטה תקבל את הנתיב לתיקייה ותנקה את תוכנה.

השיטה תוציא הודעה מתאימה במקרה של הצלחה או כישלון על פי חלוקה למקרים השונים.

```
public void showDictionary ()
```

השיטה מאפשרת את הצגת המילון על גבי ממשק המשתמש (GUI).