



1. השירות `getItemsSet` יחזיר אוסף מטיפוס `Set` אשר מכיל את כל האיברים בהיסטוגרמה ללא הספירות שלהם.

### סעיף 1 (25 נק'):

ממשו את המחלקה `HashMapHistogram` אשר מממשת את המנשק `Histogram`. המימוש יעשה באמצעות הכלה (aggregation) של `HashMap`, כלומר, כל מופע של `HashMapHistogram` יכיל שדה מטיפוס `HashMap`. שדה זה יהיה אחראי על שמירת הספירות עבור כל אובייקט. טיפוס זה הוא גנרי, כלומר, ניתן לקבוע ביצירת האובייקט מהוא טיפוס האיברים אותם יחזיק.

### סעיף 2 (25 נק'):

המנשק `Histogram` יורש מהמנשק `Iterable`, מה שמחייב את `HashMapHistogram` לממש את השירות `iterator()`.

נרצה לעבור על תוכן ההיסטוגרמה באופן הבא: נעבור על כל האיברים, החל מהאיבר עם מספר המופעים הגדול ביותר ועד לאיבר עם מספר המופעים הקטן ביותר (עבור שני איברים עם אותו מספר המופעים אין חשיבות לסדר המעבר עליהם). אפשרות אחת היא להחזיק סוג של מבנה נתונים ממויין ולתחזק את הסדר שלו בכל הכנסת איבר, אך אנחנו מעוניינים במימוש שיאפשר הכנסת איברים מהירה ולכן לא נבצע שום מיון/סידור בעת הכנסת איברים. את המחיר בביצועים נשלם כאשר נרצה לעבור על כל האיברים באמצעות האיטרטור, ואז כן יתבצע סידור.

לצורך כך עליכם לממש:

א. מחלקה חדשה המממשת את המנשק `Iterator`. ההיסטוגרמה שלנו ממומשת ע"י מיפוי (`Map`) מאיבר למספר המופעים שלו (ספירות), והאיטרטור צריך לעבור על האיברים בסדר יורד של הספירות.

ב. מחלקה חדשה המממשת `Comparator`. האיברים והספירות שלהם נמצאים במפה, כך שמיון האיברים ע"פ מספר המופעים יבוצע למעשה ע"י מיון מפתחות המפה על פי הערכים בסדר יורד. לצורך כך נשתמש במיון (`sort`) ובאמצעות `Comparator` שישווה שני איברים ע"פ ערכם במפה.

שימו לב, ניתן ואף כדאי להעביר אוספים בין המופעים של המחלקות השונות וכן להשתמש בהכלה של אוספים לפי הצורך. למשל, על מנת להשוות בין הערכים של שני מפתחות במפה, ה `Comparator` יצטרך גישה למפה עצמה. ה `iterator` בתורו יצטרך לייצר את האוסף הממויין עליו יעבור במהלך האיטרציות.

שלד כל המחלקות אותן אתם נדרשים לממש נתון לכם בחבילה `il.ac.tau.cs.sw1.ex8.histogram` המופיעה בקבצי התרגיל.

העזרו ב `HashMapHistogramTester` בשביל לבדוק את עצמכם.

## חלק ב' (50 נק')

בחלק זה נתרגל עבודה עם אוספים (Collections) ע"י מימוש מנוע פשוט להשוואה בין קבצי טקסט. בין קבצי העזר של התרגיל תוכלו למצוא את המחלקות FileUtils ו- FileSimilarityTest, ואת שלד המחלקה FileIndex (כולן בחבילה il.ac.tau.cs.sw1.ex8.filesim). יש להגיש רק את FileIndex ומחלקות עזר נוספות, אם כתבתם כאלה, בתוך תיקיות החבילה.

מנוע ההשוואה שלנו יקבל כקלט תיקיה במערכת הקבצים, יקרא את כל הקבצים בה, וישמור ב"אינדקס" עבור כל אחד מהקבצים את מספר המופעים של כל מילה (token) בו. לאחר מכן, נוכל להשוות בין הקבצים השמורים באינדקס.

סעיף 1 (20 נק'):

המתודה `index()` במחלקה `FileIndex` קוראת את הקבצים ומוסיפה אותם לאינדקס, והקוד שלה כבר נתון. השלימו את מתודות העזר בהן היא משתמשת: המתודה `clearPreviousIndex()` מנקה נתונים שנשמרו בקריאה הקודמת ל- `index()`. המתודה `addFileToIndex(File file)` מוסיפה את הנתונים הרלוונטיים לגבי קובץ טקסט יחיד לשדות המחלקה. קריאת המילים מן הקובץ תבצע בעזרת `readAllTokens(File file)` ממחלקת העזר `FileUtils`, שכבר נתונה לכם.

עבור קבצים שאינם מכילים מילים תקינות או שלא ניתן לקרוא מהם, תודפס הודעה על שגיאה והם לא יתווספו לאינדקס.

**שימו לב**, עליכם לבחור את מבני הנתונים המתאימים לייצוג המידע הדרוש (לשם כך, קראו גם את הסעיפים הבאים), תוך שימוש יעיל באוספים גנריים מתוך `Java collection framework`. בפרט, השתמשו במבנה הנתונים `HashMapHistogram` אשר מומש בחלק א' על מנת לשמור את ספירות ה-`token`ים בכל קובץ.

על מנת שלא לקשור בין המימוש הספציפי של `IHistogram` לבין המחלקה `FileIndex`, השתמשו במפעל `HistogramFactory` אשר מומש עבורכם ומופיע ב `package` של חלק זה. שימוש פרקטי של המפעל – הוא יאפשר לנו לבדוק את הקוד של חלק ב' שלכם עם מימוש תקין על `IHistogram`, במידה ויש טעויות במימוש של חלק א', כל זאת מבלי לשנות את המימוש ל `FileIndex`.

סעיף 2 (10 נק')

השלימו את מימוש המתודה `getCommonTokensNum(File file1, File file2)`. מתודה זו מקבלת כקלט שני קבצים השמורים באינדקס ומחזירה את מספר המילים המשותפות לשני הקבצים. מילה תחשב משותפת לשני הקבצים אם היא מופיעה לפחות פעם אחת בשניהם.

סעיף 3 (15 נק'):

המתודה `getCosineSimilarity(File file1, File file2)` מחזירה, עבור שני קבצים השמורים באינדקס, את ציון הדמיון שלהם `cosine similarity`, לפי הנוסחה הבאה:

$$\frac{\sum_{w \in \text{file1} \cap \text{file2}} A_w \cdot B_w}{\sqrt{\sum_{w \in \text{file1}} A_w^2} \cdot \sqrt{\sum_{w \in \text{file2}} B_w^2}}$$

דוגמא לשימוש בנוסחא: נתונים לנו 2 קבצים עם ההיסטוגרמות הבאות:

מופעים	מילה
1	love
3	my
5	guitar

מופעים	מילה
4	while
2	my
3	guitar

הדמיון ביניהם יחושב באופן הבא:

$$\frac{3 * 2 + 3 * 5}{\sqrt{4^2 + 2^2 + 3^2} * \sqrt{1^2 + 3^2 + 5^2}}$$

במונה נסכום את מכפלות המופעים של האיברים המשותפים: המילים my ו guitar. מכיוון ש love ו while מופיעות רק באחד מהקבצים, הן תורמות 0 לסכום המונה.

**הסבר לנוסחא:** אנו מסתכלים על קבצי טקסט כעל וקטורים של מילים, ומחשבים את [קוסינוס הזווית](#) ביניהם כמדד לדמיון בין הקבצים. בהינתן מילה w, נסמן ב-  $A_w$  את מספר המופעים שלה ב-file1 (כלומר, כמה פעמים היא חוזרת בתוצאת readAllTokens) וב-  $B_w$  את מספר המופעים שלה ב-file2. במונה אנחנו מחשבים את מכפלת מספרי המופעים עבור כל מילה הנמצאת בשני הקבצים, וסוכמים. לכן, אם יש הרבה מילים משותפות שחוזרות על עצמן, המונה יהיה גדול. במכנה אנחנו מחלקים [בנורמות](#) של כל אחד מהקבצים, כדי "לבטל" את השפעת גודל הקובץ ומספר החזרות של מילים בו.

בפרט, תוצאת הנוסחא היא 1 אם משוים קובץ מסוים לעצמו, ו-0 אם משוים שני קבצים שאין ביניהם מילים משותפות כלל.

השלימו את מימוש המתודה. שימו לב כי לשם כך עליכם גם להשלים את מימוש מתודת העזר `verifyFile(File file)` אשר מקבלת כקלט קובץ ומחזירה true אם זהו קובץ ששמור באינדקס. אחרת, עליה להדפיס הודעה למשתמש המתחילה ב- [ERROR] (בדומה להודעות השגיאה שמודפסות מהקוד הנתון לכם) ולהחזיר false. המתודה `getCosineSimilarity` משתמשת במתודה זו לבדיקה התחלתית של הקובץ.

סעיף 4 (5 נק')

השלימו את מימוש המתודה `getNumIndexedFiles()` המחזירה את מס' הקבצים השמורים כרגע באינדקס.

**בדקו את עצמכם:** המחלקה `FileSimilarityTest` מכילה תכנית בדיקה קצרה עבור המחלקה שכתבתם. כדי להריץ אותה, שימרו את התיקיה `testFiles` תחת תיקיית הפרויקט ב-Eclipse או תחת התיקיה ממנה אתם מריצים את קוד הג'אווה. תיקיה זו מכילה 3 קבצים. בהרצה אמור להתקבל הפלט הבא (המסלול המלא לקבצים יכול להשתנות כתלות במקום בו שמרתם את התיקיה):

```
Indexing C:\testFiles\file1.txt
Indexing C:\testFiles\file2.txt
Indexing C:\testFiles\file3.txt
Indexed 3 files.
C:\testFiles\file2.txt: cosine similarity: 0.324, number of common words: 17
C:\testFiles\file3.txt: cosine similarity: 0.503, number of common words: 20
```

התוצאה, אגב, מראה כי לפי המדד שלנו, שיר של רוברט פרוסט (ב-file1) דומה יותר לשיר אחר של אותו משורר, מאשר לשיר של שייקספיר, כפי שהיינו מצפים.

מומלץ לשנות ולהוסיף קבצים לצרכי הבדיקה.

**בהצלחה!**