

תוכנה 1 – אביב תשע"ח

תרגיל מספר 8

אוספים גנריים ו-collection framework

הנחיות כלליות:

קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

- הגשת התרגיל תיעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש aviv1 יקרא הקובץ aviv1_hw8.zip). קובץ ה-zip יכול:
 - א קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - ב תיקיה בשם hw8_files, בתוכה שתי התיקיות il, ו-resources.

הנחיות כלליות לתרגיל:

- א בכל אחד מחלקי התרגיל ניתן להוסיף שירותים ומחלקות לפי הצורך, אך אין לשנות חתימות של שירותים קיימים והגדרות של מנשקים.
- ב בכל חלק קיים טסטר קצר המבצע בדיקות שפיות. כדאי ומומלץ להוסיף בדיקות משלכם שכן הטסטרים הם בסיסיים ביותר ולא בודקים את כל המקרים.

חלק א' (50 נק')

בתרגיל זה עליכם לממש מבנה נתונים של היסטוגרמה באמצעות אוספים גנריים. נגדיר היסטוגרמה בתור מבנה נתונים אשר סופר מופעים של עצמים מטיפוס T כלשהו (טיפוס גנרי). הקוד ימומש בחבילה `il.ac.tau.cs.sw1.ex8.histogram`.

לדוגמא, עבור אוסף האיברים הבא: 1, 2, 3, 1, 2, 1, 3, 2, 1, 3, 2, 1 ואת מספר המופעים שלהם.

יחד עם קבצי התרגיל מסופק לכם הממשק IHistogram המכיל שישה שירותים:

```
public interface IHistogram<T> extends Iterable<T> {
    public void addItem(T item);
    public void removeItem(T item);
    public void addItemKTimes(T item, int k) throws IllegalArgumentException;
    public void removeItemKTimes(T item, int k) throws IllegalArgumentException;
    public void addAll(Collection<T> items);
    public int getCountForItem(T item);
    public void clear();
    public Set<T> getItemsSet();
}
```

א. השירות addItem מוסיף מופע אחד של הפריט item להיסטוגרמה.

- ב. השירות `removeItem` מוריד מופע אחד של הפריט `item`. אם הפריט לא נמצא בהיסטוגרמה, הפונקציה תזרוק את החריג `IllegalItem` שמימושו נתון לכם.
- ג. השירות `addItemKTimes` מוסיף `k` מופעים של הפריט `item`. עבור `k` קטן מ-0 הפונקציה תזרוק את החריג `IllegalKValue` שמימושו נתון לכם.
- ד. השירות `removeItemKTimes` מוריד `k` מופעים של הפריט `item`. עבור `k` גדול ממספר המופעים של הפריט הפונקציה תזרוק את החריג `IllegalKValue` שמימושו נתון לכם.
- ה. השירות `addAll` מוסיף אוסף של פריטים להיסטוגרמה.
- ו. השירות `getItemCount` יחזיר את מספר הפעמים שהאיבר `item` נספר. אם `item` הוא פריט שלא קיים בהיסטוגרמה, יוחזר הערך 0.
- ז. השירות `clear` ירוקן את ההיסטוגרמה מכל האיברים והספירות (כלומר, לאחר `clear`, השירות `getItemCount` יחזיר ספירה 0 לכל איבר).
- ח. השירות `getItemSet` יחזיר אוסף מטיפוס `Set` אשר מכיל את כל האיברים בהיסטוגרמה אשר מספר המופעים שלהם גדול מ-0, ללא הספירות שלהם.

סעיף 1 (25 נק'):

ממשו את המחלקה `HashMapHistogram` אשר ממששת את הממשק `IHistogram` עבור כל טיפוס `T` המממש את הממשק `Comparable` (כלומר, `T` יכול לקבל ערך של כל מחלקה המממשת את הממשק `Comparable`). נזכיר כי הטיפוסים המובנים הבסיסיים כמו `Integer` ו `String` מממשים ממשק זה). לדרישה הזו יש סיבה אותה ניראה בהמשך.

פרקטית, זה אומר שנגדיר את `HashMapHistogram` באופן הבא:

```
public class HashMapHistogram<T extends Comparable<T>> implements
    IHistogram<T>
```

משמעות הגדרה הזו: הפרמטר הגנרי למחלקה `HashMapHistogram` הוא `T` אשר מממש את הממשק `Comparable`. `Comparable` הוא בעצמו גנרי, והפרמטר שנעביר לו הוא `T`: כלומר, אנחנו דורשים איברים מטיפוס `T` אשר ניתנים להשוואה עם איברים מטיפוס `T`, כלומר, מאותו הטיפוס.

פרמטר `T` זה הוא גם הפרמטר שמצריך הממשק `IHistogram` (ששם אין שום הגבלה על הפרמטר הגנרי).

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 מטיפוס `T`.

סעיף 2 (25 נק'):

הממשק `IHistogram` יורש מהממשק `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.

לצורך כך עליכם לממש:

- א. מחלקה חדשה המממשת את הממשק `Iterator`. ההיסטוגרמה שלנו ממומשת ע"י מיפוי (`Map`) מאיבר למספר המופעים שלו (ספירות), והאיטרטור צריך לעבור על האיברים בסדר הבא:

a. נעבור על האיברים בסדר יורד של הספירות: כלומר, האיבר הראשון שיוחזר הוא האיבר בעל מספר הספירות המקסימלי.

b. עבור שני איברים בעלי אותו מספר מופעים נבצע שבירת שוויון באמצעות השוואת האיברים עצמם. השוואה זו אפשרית רק בגלל שדרשנו שההיסטוגרמה תחזיק איברים בעלי השוואה (Comparable) בינם לבין עצמם. עבור שני איברים עם מספר מופעים זהה נחזיר קודם את האיבר הקטן יותר על פי הסידור הטבעי של האיברים. לדוגמא: אם האיברים שלי הם המספרים 1 ו 3, ושניהם נספרו אותו מספר פעמים, קודם יחזור 1 ואחריו 3.

אין צורך לממש את פעולת ה remove.

b. מחלקת Comparator. האיברים והספירות שלהם נמצאים במפה, כך שמיון האיברים ע"פ מספר המופעים יבוצע ע"י מיון הערכים (ספירות) בסדר יורד. אל תשכחו לטפל במקרה של שוויון בספירות. לצורך כך נשתמש במיון (sort) ובאמצעות Comparator שישווה שני איברים ע"פ הקריטריונים שהוגדרו בתת הסעיף הקודם. ניתן לממש מחלקה זו כמחלקה פנימית במחלקת האיטרטור או כמחלקה בקובץ Java נפרד משלה.

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. האיטרטור 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, והוסיפו לו בדיקות משלכם.

חלק ב' (50 נק')

בחלק זה נתרגל עבודה עם אוספים (Collections) ע"י מימוש מנוע אשר אוסף סטטיסטיקות על מילים בקבצי טקסט ומדרג אותן לפי קריטריונים שונים. חלק מהמשימות בתרגיל זה מוכרות לכם מתרגילים קודמים, אך עכשיו יש בידינו כלים המאפשרים לנו לבצע אותן ביעילות רבה יותר.

הקוד בחלק זה ימומש בחבילה `il.ac.tau.cs.sw1.ex8.wordsRank` אך ישתמש גם בקוד של ההיסטוגרמה אותה מימשתם בחלק ב' (כלומר, ישתמש בקוד שמופיע בחבילה אחרת – וודאו ששני החלקים האלה מופיעים אצלם באותו הפרוייקט ב eclipse)

מנוע הדירוג שלנו יקבל כקלט תיקיה במערכת הקבצים, יקרא את כל הקבצים בה, ויבצע פעולת אינדקס שבה ישמרו כל הספירות הרלוונטיות לפעולות אותה המנגנון צריך לספק.

סעיף 1

המתודה `indexDirectory()` במחלקה `FileIndex` קוראת את הקבצים ומוסיפה אותם לאינדקס. המימוש של פונקציה זו נתון לכם חלקית ואתם רשאים לערוך אותו. קריאת המילים מן הקובץ תתבצע בעזרת `readAllTokens(File file)` ממחלקת העזר `FileUtils`, שכבר נתונה לכם.

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, עליכם להשתמש במבנה הנתונים HashMapHistogram אשר מומש בחלק א' על מנת לשמור את ספירות ה-tokenים בכל קובץ.

הערות נוספות:

- שם תיקיית הקבצים יהיה שם חוקי של תיקיה המכילה לפחות קובץ אחד.
- השירות readAllTokens של FileUtils מבטל סימני פיסוק ומחזיר מילים שאינן ריקות, אין לבצע עיבוד או סינון נוסף בגוף המימוש שלכם: כל המילים שחוזרות ע"י readAllTokens הן חוקיות מבחינתכם.
- הניחו כי פעולת האינדקס תבוצע פעם אחת בלבד על כל אובייקט מטיפוס FileIndex.

סעיף 2

ממשו את השירות getCountInFile במחלקה FileIndex אשר מקבל מחרוזת filename ומחרוזת word אשר מחזיר את מספר המופעים של המילה word בקובץ filename. עבור מילה שאינה מופיעה בקובץ יוחזר הערך 0.

הנחיות כלליות לסעיף זה והסעיפים אחריו:

- בכל שירות המקבל שם של קובץ, המחרוזת filename מכילה שם קובץ בלבד (ללא נתיב), ויש לחפש אותו בתיקיה עליה בוצע שלב ה index (ראו דוגמת שימוש במחלקת הטסטר).
- בכל שירות המקבל שם של קובץ, במידה ושם הקובץ אינו קיים בתיקיה זו, יש לזרוק חריג מטיפוס FileNotFoundException (מומש עבורכם) עם הודעה אינפורמטיבית לבחירתכם.
- בכל שירות שמקבל מילה word יש להמירה ל lowercase לצורך ביצוע החיפוש באינדקס.

חתימת השירות:

```
public int getCountInFile(String filename, String word) throws  
FileIndexException
```

סעיף 3

נגדיר את המושג "דרגה" (rank) עבור מילה בקובץ. דרגתה של המילה word היא מיקום המילה ברשימה הממויינת של כל המילים בקובץ על פי השכיחות שלהם (בסדר יורד). עבור שתי מילים עם אותו מספר מופעים, נסדר את הדרגות לפי סדר לקסיקוגרפי (הסדר הטבעי של מחרוזות). רמז: זה בדיוק הסדר שבו עובר האיטורטור של היסטוגרמה.

לדוגמא, עבור קובץ המיוצג ע"י ההיסטוגרמה הבאה: "I": 7, "me":3, "mine":4, "all":5, הדרגה של המילה "I" היא 1, הדרגה של המילה "all" היא 2, הדרגה של המילה "mine" היא 3, והדרגה של המילה "me" היא 4 (שימו לב שהדרגה הראשונה היא תמיד 1, לא 0).

אנחנו מעוניינים לבחון דרגות של מילים בכמה קבצים שונים ולבצע השוואות ביניהם. לצורך כך, ניתן לכם המימוש של המחלקה RankedWord. מחלקה זו שומרת את הדרגות של מילה כלשהי בכל הקבצים באינדקס, ובנוסף, שומרת את הדרגה המינימלית, המקסימלית והממוצעת על פני כל הקבצים.

לדוגמא: נניח כי עבור המילה "all" דרגתה בקובץ הראשון היא 3, בקובץ השני 5 ובקובץ השלישי 4. הדרגה המינימלית שלה היא 3, הדרגה המקסימלית שלה הוא 5, והדרגה הממוצעת על פני שלושת הקבצים היא $(3+4+5)/3$.

השלימו את מימוש המחלקה RankedWordComparator אשר מאפשר השוואה בין איברים מטיפוס RankedWord לפי אחת משלוש אופציות: דרגה מקסימלית, מינימלית וממוצעת. אופן ההשוואה נקבע בבנאי של comparator זה. איבר x נחשב "קטן" יותר מאיבר y אם הדרגה הרלוונטית (למשל, דרגה מקסימלית) של x קטנה יותר מזו של y. (כאשר שתי הדרגות זהות, אין חשיבות לסדר בין האיברים).

הערה: המחלקה RankedWord וה Comparator שמימשתם לה הן מחלקות שימושיות מאוד עבור התרגיל. אתם לא מחוייבים להשתמש בהן, אבל זה מאוד מומלץ.

סעיף 4

ממשו את שירות getRankForWordInFile במחלקה FileIndex אשר מקבל מחרוזת filename ומחרוזת word ומחזיר את הדרגה של word בקובץ filename. במידה והמילה אינה מופיעה באותו הקובץ, יש להחזיר את מספר 0. טיפול זה במילים שאינן מופיעות בקובץ תקף גם לשאר הסעיפים בתרגיל.

חתימת השירות:

```
public int getRankForWordInFile(String filename, String word) throws  
FileIndexException
```

רמז: הגדרת מבני נתונים נכונים ובנייתם בשלב האינדקס תפשט מאוד את המימוש של שירות זה והשירות בסעיף 6 לכדי שליפה מתוך מבנה נתונים.

סעיף 5

ממשו את השירות getAverageRankForWord אשר מקבל מחרוזת word ומחזיר את הדירוג הממוצע של המילה word על פני כל הקבצים באינדקס. הפונק' צריכה להחזיר ערך גם אם word לא נראתה באף אחד מהקבצים באינדקס, ע"י אופן החישוב המפורט בסעיף הקודם. שימו לב: אם אתם משתמשים במחלקה RankedWord אין לכם צורך לבצע בעצמכם את חישוב הממוצע.

חתימת השירות:

```
public int getAverageRankForWord(String word)
```

סעיף 6

ממשו את שלושת השירותים הבאים:

```
public List<String> getWordsWithAverageRankLowerThenK (int k)
```

```
public List<String> getWordsBelowMinRank(int k)
```

```
public List<String> getWordsAboveMaxRank(int k)
```

השירות `getWordsWithAverageRankLowerThenK` יחזיר את כל המילים להן דרגה ממוצעת קטנה או שווה ל `k`. המילים יהיו ממויינות בסדר עולה ע"פ קריטריון זה (כלומר, נתחיל מהמילה עם הדרגה ממוצעת הכי קטנה, וכן הלאה – אם ישנם שני איברים ודרגתם זהה, אין חשיבות לסדר ביניהם).

באופן דומה, שני השירותים האחרים יבצעו מיון בסדר עולה על פי דרגה מינימלית ודרגה מקסימלית. גם בסעיף זה, כמו בסעיף 6, עליכם לתמוך במילים שלא הופיעו באף קובץ.

את פעולת המיון ע"פ שלושת הקריטריונים נרצה לבצע רק על פי הצורך, כלומר, לא בשלב האינדקס, שכן יתכן ולא נשתמש בשירותים אלה כלל במהלך ריצת התוכנית. ניתן לשמור את מבנה הנתונים ולמיין אותו בכל פעם בהתאם לצורך, או לחילופין, לייצר בכל פעם מבנה נתונים עליו יבוצע המיון (יש יתרונות וחסרונות בשתי הגישות).

רמז: חישובו על פונקציית עזר בה יכולות להשתמש שלושת הפונקציות האלה. כמו בסעיף הקודם, אם האינדקס שלכם בנוי נכון, לא תצטרכו לבצע חישובי דרגות אלא להשתמש בחישובים קיימים.

טסטר:

בדקו את עצמכם באמצעות `FileIndexTester`. עדכנו את הקבוע `TEST_FOLDER` על פי מיקום התיקיה `resources` אצלכם על המחשב.

הטסטר מכיל שתי בדיקות: בדיקה של ה `Comparator` עבור `RankedWord` ובדיקת ה `FileIndex`.

כמו בכל תרגיל אחד, הטסטר הוא טסטר בסיסי שאינו בודק את כל המקרים, וריצה מוצלחת שלו מהווה תנאי הכרחי אך לא מספיק בשביל לוודא שהתרגיל שלכם עובד כנדרש. הוסיפו בדיקות משלכם!

בהצלחה!