תוכנה 1 – חורף 2020/21

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 (http://moodle.tau.ac.il/), וניתן להגיש בנוסף להגשה במודל גם בגיט.
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש stav1 יקרא הקובץ stav1_hw8.zip). קובץ ה-zip יכיל:
  א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
  ב. 2 התיקיות src וresources.

  יש להגיש את הzip ואת מבנה התיקיות שבתוכו בדיוק באותה היררכיה שקיבלת אותם.
- להגשה בגיט צרו את הrepository שלכם מתוך הקישור הבא:
  https://classroom.github.com/a/CtsEQPgR

הנחיות כלליות לתרגיל:

א. בכל אחד מחלקי התרגיל ניתן להוסיף שירותים ומחלקות לפי הצורך, אך אין לשנות חתימות של שירותים קיימים והגדרות של מנשקים (גם אין להוסיף או להוריד throws).

ב. אין הגבלות לimportים או שימוש במבני נתונים מסוימים, אך שימו לב שאתם מוחקים importים מיותרים או כאלה שאינם מתקמפלים בנובה.

ג. בכל חלק קיים טסטר קצר המבצע בדיקות שפיות (חפשו בגוגל sanity tests). 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, ההיסטוגרמה תכיל את האיברים 1, 2, 3 ואת מספר המופעים שלהם.

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

```
public void addItem(T item);
public void removeItem(T item) throws IllegalItemException;
public void addItemKTimes(T item, int k) throws IllegalKValueException;
public void removeItemKTimes(T item, int k) throws IllegalItemException,
IllegalKValueException;
public int getCountForItem(T item);
public void addAll(Collection<T> items);
public void clear();
public Set<T> getItemsSet();
public void update(IHistogram<T> anotherHistogram);
```

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

ב. השירות removeItem מוריד מופע אחד של הפריט item. אם הפריט לא נמצא בהיסטוגרמה (מספר המופעים שלו הוא 0), הפונקציה תזרוק את החריג `IllegalItemException` שמימושו נתון לכם.

ג. השירות `addItemKTimes` מוסיף k מופעים של הפריט item. עבור k קטן מ-1 הפונקציה תזרוק את החריג `IllegalKValueException` שמימושו נתון לכם.

ד. השירות `removeItemKTimes` מוריד k מופעים של הפריט item. אם הפריט לא נמצא בהיסטוגרמה, הפונקציה תזרוק את החריג `IllegalItemException`. אם הפריט נמצא ו k גדול ממספר המופעים של הפריט או k קטן מ-1 הפונקציה תזרוק את החריג `IllegalKValueException` שמימושו נתון לכם.

ה. השירות `addAll` מוסיף אוסף של פריטים להיסטוגרמה. (לא ניתן להניח שכל האיברים נמצאים כבר בהיסטוגרמה).

ו. השירות getCountForItem 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 `getCountForItem` יחזיר ספירה 0 לכל איבר).

ח. השירות `getItemsSet` יחזיר אוסף מטיפוס Set אשר מכיל את כל האיברים בהיסטוגרמה אשר מספר המופעים שלהם גדול מ-0, ללא הספירות שלהם.

ט. השירות update מעדכן את ההיסטוגרמה עם כל הפריטים ומספר המופעים שלהם ב `anotherHistogram`. אם פריט קיים בהיסטגורמה אז יש להוסיף למספר המופעים שלו את מספר המופעים שיש ב`anotherHistogram`.

<u>סעיף 1 (25 נק'):</u>

ממשו את המחלקה HashMapHistogram אשר מממשת את המנשק IHistogram עבור כל טיפוס T המממש את המנשק Comparable (כלומר, T יכול לקבל ערך של כל מחלקה המממשת את המנשק Comparable. נזכיר כי הטיפוסים המובנים הבסיסיים כמו Integer ו String מממשים מנשק 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: המנשק IHistogram מצריך פרמטר גנרי. את הפרמטר הגנרי נגדיר כ T, ונוסיף עליו את האילוץ שהוא צריך לממש את המנשק Comparable<T>. כלומר, T יהיה פרמטר גנרי מתאים אם הוא Comparable עם עצמים אחרים מטיפוס T.

המימוש יעשה באמצעות הכלה (aggragation) של HashMap, כלומר, כל מופע של HashMapHistogram יכיל שדה מטיפוס HashMap. שדה זה יהיה אחראי על שמירת הספירות עבור כל אובייקט מטיפוס T.

<u>סעיף 2 (25 נק')</u>

המנשק IHistogram 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.

לצורך כך עליכם לממש:

א. מחלקה חדשה המממשת את המנשק Iterator. ההיסטוגרמה שלנו ממומשת ע"י מיפוי (Map) מאיבר למספר המופעים שלו (ספירות), וה Iterator צריך לעבור על האיברים בסדר הבא:
   a. נעבור על האיברים בסדר יורד של הספירות: כלומר, האיבר הראשון שיוחזר הוא האיבר בעל מספר המופעים 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.

ב. מחלקת Comparator. האיברים ומספר המופעים שלהם נמצאים במפה, כך שמיון האיברים ע"פ מספר המופעים יבוצע ע"י מיון הערכים (ספירות) בסדר יורד. אל תשכחו לטפל במקרה של שוויון במספר המופעים. 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 למחלקות ה-Comparator וה-Iterator את המידע הרלוונטי מתוך המופע של הHashMapHistogram 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. להבין איזה מידע כל אובייקט צריך לקבל הוא חלק מהאתגר מהתרגיל, חישבו על כך בעת בניית המחלקות.

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 א' (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 שהמחלקה זורקת שגיאה ואין לשנות זאת, ניתן במקום זאת להיעזר בtry וcatch אם יש צורך בכך.

המטרה של שירות זה היא לקרוא את תוכן כל הקבצים, לנתח אותו ולשמור אותו כך שהמימוש של שאר השירותים במחלקה FileIndex יהיה יעיל ומהיר. בתרגיל זה, השאיפה שלנו היא שזמן הביצוע של הפעולות ב FileIndex יהיה נמוך ככל האפשר, גם במחיר שפעולת ה index תהיה כבדה, כלומר, תבצע הרבה חישובים ותשמור מבני נתונים שונים. הגדרה נכונה של מבני הנתונים, והוצאת קוד משותף למתודות פרטיות תהפוך את המימוש של חלק מהשירותים המוגדרים ב fileIndex לפעולות שליפה פשוטות ממבני נתונים. אתם לא נמדדים על זמן הריצה של המתודות, אך זוהי שיטת העבודה הרצויה.

שימו לב, עליכם לבחור את מבני הנתונים המתאימים לייצוג המידע הדרוש. לשם כך, **עליכם לקרוא ולהבין את כל הסעיפים של חלק ב'**, ורק לאחר מכן לקבל את ההחלטה על מבני הנתונים שישמשו אתכם. עליכם להשתמש ביעילות במבני נתונים גנריים מתוך Java collection framework. בפרט, עליכם להשתמש במבנה הנתונים HashMapHistogram אשר מומש בחלק א' על מנת לשמור את מספר המופעים של ה token-ים 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.
- קובץ יכול להיות קובץ ריק מתוכן או ללא מילים חוקיות.
- 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 (שימו לב שזה בשונה מ getRankForWordInFile כפי שמוגדר בסעיף 4 והלאה).

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). כמובן שלאחר שבוצע האינדוקס, אפשר להסתפק בבדיקה באינדקס, ואין צורך לקרוא מהדיסק.
- בכל שירות המקבל שם של קובץ, במידה ושם הקובץ אינו קיים בתיקיה זו, יש לזרוק חריג מטיפוס `FileIndexException` (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
```

<u>סעיף 3</u>

נגדיר את המושג "דרגה" (`rank`) עבור מילה בקובץ. דרגתה של המילה `word` היא מיקום המילה ברשימה הממויינת של כל המילים בקובץ על פי השכיחות שלהן (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,
```

אם נמיין את המילים בסדר יורד של מספר המופעים שלהן, נקבל את הסדר הבא: `I`, אחריה המילה `all`, אחריה המילה `mine` ולבסוף `me`. לכן, 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).

אנחנו מעוניינים לבחון דרגות של מילים בכמה קבצים שונים ולבצע השוואות ביניהן. לצורך כך, נשתמש במחלקה `RankedWord`. מופע של `RankedWord` שומר עבור מילה מסויימת את הדרגות שלה 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, שני האיברים נחשבים לזהים).

השתמשו ב RankedWordComparatorTester על מנת לבדוק את המימוש שלכם.

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 ערך השווה למספר המילים השונות בקובץ + 30 (השתמשו בקבוע UNRANKED_CONST). טיפול זה במילים שאינן מופיעות בקובץ תקף גם לשאר הסעיפים הבאים בתרגיל.

חתימת השירות:

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

סעיף 5

ממשו את השירות getAverageRankForWord אשר מקבל מחרוזת word ומחזיר את הדירוג הממוצע של המילה word על פני כל הקבצים באינדקס. השירות יחזיר ערך גם אם המילה לא ניראתה כלל באף קובץ (כלומר ממוצע הדרגות על פני כל הקבצים, כאשר הדרגה בכל קובץ היא מספר המילים השונות בקובץ + UNRANKED_CONST). חישוב הממוצע יעשה על פני כל הקבצים, גם אלו שהמילה הופיעה בהם וגם אלו שלא.
שימו לב: שימוש במחלקה RankedWord יחסוך לכם את את המימוש של חישוב הדרגה המבוצעת, זאת כיוון שהמחלקה מבצעת זאת בעצמה.

חתימת השירות:

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

סעיף 6

ממשו את שלושת השירותים הבאים:

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

public List<String> getWordsWithMinRankSmallerThanK(int k)

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

השירות `getWordsWithAverageRankSmallerThanK` יחזיר את כל המילים להן דרגה ממוצעת קטנה ממש 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. עליכם לתמוך במילים שהופיעו רק בחלק מהקבצים. גם כאן, יש להתחשב בדירוג המילה גם בקבצים שהמילה הופיעה בהם וגם בכאלה שלא.

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: חישבו 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.

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

עבור חישובי הדרגות למילים עבור מילים שלא הופיעו בקובץ יש להשתמש בהגדרה מסעיף 4 (מספר המילים השונות בקובץ + 30).

הסבר קצר על השימוש ב enum:

בתרגיל זה, הקוד כולל שימוש ב enum, סוג מחלקה עליה תלמדו בהמשך הקורס. במחלקה `RankedWord` מוגדר עבורכם ה enum ששמו `rankType`. זהו למעשה אוסף של שלושה קבועים אפשריים המייצגים 3 סוגים של דרגות משוקללות על פני כל הקבצים: דרגה מינימלית, מקסימלית וממוצעת. לשלושת הקבועים האלה יש טיפוס אחד שמאחד אותם, מה שמאפשר לנו להגדיר שדות ומשתנים מטיפוס זה (הטיפוס הוא `rankType`).

מימוש אפשרי אחר (ללא enum) היה להשתמש במשתנה מטיפוס int ולשלוח כל פעם אחד משלושה ערכים שיוקצו לכל אופציה (למשל, 0 למינימום, 1 למקסימום ו 2 לממוצע), אבל יש לזה חסרונות עליהם תדברו בהרצאה.

מבחינת השימוש ב enum, אלה הן שתי פעולות שיכולות להיות שימושיות עבורכם:

1. העבר ערך כפרמטר. זה נעשה בצורה הבא:

```
rWord.getRankByType(rankType.min)
```

בדוגמא זו יש לנו משתנה rWord מטיפוס RankedWord. נרצה לשלוף את הדרגה המינימלית שלה, ולכן נשלח את הקבוע rankType.min.

2. בדיקת ערך של פרמטר:

```
if (rType == rankType.min)
```

בדוגמא זו יש לנו המשתנה rType מטיפוס rankType, ונרצה לבדוק אם הערך שלו הוא rankType.min.

ניתן לשאול שאלות נוספות בפורום התרגיל, וכאמור, החומר ילמד בהרצאה הקרובה. מומלץ גם לחפש בגוגל על הטיפוס אם ישנה אי בהירות לגבי ההתנהגות שלו.

<u>טסטרים:</u>

כמו בכל תרגיל אחד, הטסטרים הם טסטרים בסיסיים שאינם בודקים את כל המקרים, וריצה מוצלחת שלהם מהווה תנאי הכרחי אך לא מספיק בשביל לוודא שהתרגיל שלכם עובד כנדרש. הוסיפו בדיקות משלכם!

בהצלחה!