תוכנה 1 – חורף תשע"ח

תרגיל מספר 8

BufferedWriter , 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 יכיל:
  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' (15 נק')

בתרגיל זה נממש גירסא בסיסית של המחלקה BufferedWriter עליה למדתם בתרגול 5 (כדאי לחזור על התרגול). הקוד ימומש בחבילה `il.ac.tau.cs.sw1.ex8.bufferedIO`.

כזכור, העקרון המנחה של מחלקה זו הוא שהיא עוטפת זרמים אחרים (לרוב FileWriter) ודרכם כותבת מספר קבוע של תוים לקובץ, באופן שקוף למשתמש (ה BufferedReader עובד באופן דומה). בכל פעולת כתיבה דרך ה BufferedWriter, כתיבה לקובץ מתבצעת רק אם ה buffer של ה BufferedWriter מלא.

<u>המחלקה `MyBufferedWriter`</u>:

מחלקה זו מממשת את המנשק `IBufferedWriter` (מוגדר עבורכם בחבילה bufferedIO).

בנאי המחלקה מקבל:

- fWriter -אובייקט מטיפוס `FileWriter`. להזכירכם, ה `FileWriter` מאפשר כתיבה של מחרוזות/מערכי תוים לקובץ.
- `bufferSize` - מספר שלם וחיובי (גדול מ-0) – גודל ה `buffer`.
- הבנאי יאתחל buffer שיכיל את התוים שטרם נכתבו לקובץ.

המתודה `write`:

- מקבלת מחרוזת `str` שאמורה להיכתב לקובץ ע"י שימוש ב `FileWriter`. בכל גישה ל `FileWriter` עלינו לכתוב מספר תוים ששווה לערכו של `bufferSize` (שאותחל בבנאי).
- מתודה זו תחליט על מספר הכתיבות לקובץ על סמך `bufferSize`, תוכן ה `buffer` הנוכחי וכן אורך המחרוזת `str` שהתקבלה כפרמטר ל `write`.
  - לדוגמא: אם גודל ה `buffer` הוא 5 תוים ונרצה לכתוב מחרוזת בעלת 7 תווים, 5 תווים יכתבו לקובץ, ו2 תווים ישמרו ב `buffer`. בפעולת ה `write` הבאה, אם נרצה לכתוב 2 תווים, הם יצטרפו ל 2 התווים שכבר היו ב `buffer` והוא יכיל 4 תווים שלא נכתבו לקובץ.

המתודה `close`:

- יש לסגור את ה `FileWriter` במתודה זו. בנוסף, במידה וקיימים תוים ב `buffer` שטרם נכתבו, יש לכתוב אותם בפונק' זו.

<u>בדקו את עצמכם:</u>

הטסטר של חלק זה הוא מאוד בסיסי ובודק את התוכן הנקרא, אך לא את השימוש ב buffer. חלק מהעבודה שלכם היא לתכנן בדיקה שתוכל לבדוק גם את פעולת ה buffer (כלומר, שאתם משתמשים ב FileWriter רק כאשר ה buffer מצריך זאת, ולא בכל פעולת write של המשתמש).

<u>הנחיה:</u> השתמשו במחלקה `MyFileWriter` אשר מימושה מופיע בחבילת התרגיל. מחלקה זו יכולה לסייע לכם לנתר את מספר הכתיבות שנעשו לקובץ מחוץ למימוש של `MyBufferWriter`. מחלקה בנויה על פי design pattern שנקרא decorator והוא מאוד שימוש בבעיות רבות, כמו גם בתרגיל שלנו. ע"י מעבר על מימוש המחלקה הסיקו כיצד ניתן לשלבה בתסריט הבדיקות שלכם.

<u>הנחות והנחיות נוספות:</u>

א. הניחו כי הכותב שמתקבל כפרמטר בבנאי אינו null ואינו סגור.

ב. <u>אין לייצר ולהשתמש בשום Stream למעט זה שמתקבל ע"י הבנאי של הכותב</u>. שימו לב שאתם לא מקבלים את שם הקובץ ממנו אתם קוראים כך שכל העבודה מתבצעת ע"י הפעלת ה FileWriter שקיבלתם בבנאי. כמות המידע שתיכתב בכל פעם תלויה בגודל ה buffer שגם הוא מאותחל בבנאי.

ג. השתדלו לצמצם את מספר המחרוזות הביניים הנוצרות בריצה אחת של המתודה write. מימוש טוב לא ייצר יותר משתי מחרוזות ביניים בריצה אחת של write.

ד. מבנה הנתונים של ה buffer נתון לשיקולכם. מי שרוצה לכתוב מימוש כמה שיותר קרוב למימוש האמיתי של ה BufferedWriter יכול לנסות ולממש אותו באמצעות מערך של תוים (char[]).

ה. **<u>בחלק זה בלבד</u>**, אין להשתמש באוספים גנריים.

חלק ב' (35 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 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 IllegalKValue;
	public void removeItemKTimes(T item, int k) throws IllegalKValue;
	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, או מספר המופעים שלו כבר 0, אז השירות לא עושה דבר.
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 getCountForItem יחזיר את מספר הפעמים שהאיבר item נספר. אם item הוא פריט שלא ראינו כלל, 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 ללא הספירות שלהם.

<u>סעיף 1 (20 נק'):</u>

ממשו את המחלקה HashMapHistogram אשר ממששת את המנשק IHistogram עבור כל טיפוס T הממשש את המנשק Comparable (כלומר, T יכול לקבל ערך של כל מחלקה הממשת 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: הפרמטר הגנרי למחלקה HashMapHistogram הוא T אשר מממש את המנשק 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 המנשק IHistogram (ששם אין שום הגבלה על הפרמטר הגנרי).

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 (15 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), 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.

ב. 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 ג' (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.

<u>סעיף 1 (20 נק')</u>

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, <u>עליכם להשתמש במבנה הנתונים HashMapHistogram</u> אשר מומש בחלק א' על מנת לשמור את ספירות 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.
- הניחו כי פעולת האינדקס תבוצע פעם אחת בלבד על כל אובייקט מטיפוס `FileIndex`.

<u>סעיף 3 (5 נק')</u>

ממשו את השירות `getCountInFile` 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 `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
```

סעיף 4 (5 נק')

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: 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 5 (5 נק')

ממשו את שירות getRankForWordInFile אשר מקבל מחרוזת filename ומחרוזת word ומחזיר את הדרגה של word בקובץ filename. במידה והמילה אינה מופיעה באותו הקובץ, יש להחזיר את מספר המילים השונות בקובץ זה + הקבוע UNRANKED אשר מוגדר עבורכם. (למשל, אם בקובץ זה יש 200 מילים שונות והקבוע UNRANKED שווה ל 20, יוחזר הערך 220). 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 6 (5 נק')

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 ניראתה 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 7 (10 נק')

ממשו את שלושת השירותים הבאים:

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

public List<String> getWordsBelowMinRank(int k)

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

השירות getWordsBelowAverageRank 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 בכל הקבצים.

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.

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, חומר ילמד בהרצאה הקרובה.

<u>טסטר:</u>

בדקו את עצכמם 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!