

תוכנה 1 – אביב תשע"ה

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 aviv יקרא הקובץ aviv_hw8.zip). קובץ ה-zip יכול:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - ב. קבצי ה-java של התוכניות אותם התבקשתם לממש.

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 כל מני מקרי קצה.
- ג. יש להגיש את התרגיל באופן דומה להגשת התרגילים הקודמים – קובץ zip אשר מכיל את כל התיקיות מתחת ל src.

חלק א' (15 נק')

בתרגיל זה עליכם לממש את האיטרטור Steplerator אשר יודע לעבור על איברי אוסף (Collection) כלשהו בדילוג קבוע, החל מהאיבר הראשון. הבנאי של המחלקה מקבל שני פרמטרים: collection – אוסף גנרי כלשהו, step i – משתנה מטיפוס int. האיטרטור יעבור על האוסף collection על פי הסידור הקיים שלו באופן הבא: הוא יתחיל באיבר הראשון, לאחר מכן ידלג על step-1 איברים וידפיס את האיבר הבא.

דוגמא: עבור רשימה המכילה את האיברים 1,2,3,4,5,6,7,8,9,10, איטרטור אשר עובר בדילוגים של 3 יחזיר את האיברים 1,4,7,10. על אותה הרשימה, דילוגים של 5 יחזירו את האיברים 1,6.

במידה והבנאי מקבל step שהוא 0 או שלילי, על הבנאי לזרוק את החרגי IllegalStepException (מומש עבורכם ומופיע יחד עם קבצי התרגיל).

הערות נוספות למימוש:

- אין להניח שום הנחות לגבי ה Collection שה Steplerator מקבל בבנאי, בפרט, אין להניח שהאוסף הוא סוג של רשימה (List).
- ניתן להניח שלא יעשו שינויים באוסף מרגע יצירת האיטרטור ועד סיום השימוש בו.
- אין למיין את האוספים, עליכם לעבור עליהם על פי הסידור הקיים שלהם, תוך שימוש באיטרטור של כל אוסף.

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 מופעים אין חשיבות לסדר המעבר.

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 יורד של הספירות. אין צורך לממש את פעולת ה remove.
- ב. מחלקה חדשה המממשת Comparator. האיברים והספירות שלהם נמצאים במפה, כך שמיון האיברים ע"פ מספר המופעים יבוצע ע"י מיון הערכים (ספירות) 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' (45 נק')

בחלק זה נתרגל עבודה עם אוספים (Collections) ע"י מימוש מנוע אשר אוסף סטטיסטיקות על מילים בקבצי טקסט ומדרג אותן לפי קריטריונים שונים.

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 (20 נק')

המתודה index() במחלקה FileIndex קוראת את הקבצים ומוסיפה אותם לאינדקס. המימוש של פונקציה זו נתון לכם חלקית ואתם רשאים לערוך אותו. קריאת המילים מן הקובץ תבצע 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, השתמשו במבנה הנתונים HashMapHistogram 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`.

סעיף 3 (5 נק')

ממשו את השירות `getCountInFile(String filename, String 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, המחזורת `filename` מכילה שם קובץ בלבד (ללא נתיב), ויש לחפש אותו בתיקה עליה בוצע שלב ה `index` (ראו דוגמת שימוש במחלקת הטסטר).
- בכל שירות המקבל שם של קובץ, במידה ושם הקובץ אינו קיים, יש לזרוק חריג מטיפוס `FileNotFoundException` (מומש עבורכם) עם הודעה אינפורמטיבית לבחירתכם.
- את המילה `word` שמקבל כל שירות יש להמיר ל `lowercase` לצורך ביצוע החיפוש.

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).

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).

עבור שתי מילים להן אותו מספר מופעים, סדר הדרגות אינו משנה. כך למשל, אם ל `me` היו 4 מופעים, `me` יכלה לקבל את הדרגה 3 ו `mine` את הדרגה 4, או להיפך (מבחינה פרקטית, בקבצים גדולים יחסית נדיר למצוא 2 מילים עם מספר מופעים זהה, בעיקר כיוון שנוטים להתעלם מילים אשר מופיעות מספר מועט של פעמים).

ממשו את השירות `getRankForWordInFile(String filename, String word)` אשר מחזירה את הדרגה של `word` בקובץ `filename`. מידה והמילה אינה מופיעה באותו הקובץ, יש להחזיר את מספר המילים בקובץ + הקבוע `UNRANKED` אשר מוגדר עבורכם. (למשל, אם בקובץ מספר המילים הוא 200 והקבוע `UNRANKED` שווה ל 10, יוחזר הערך 210). טיפול זה במילים שאינן מופיעות תקף גם לשאר הסעיפים בתרגיל.

סעיף 5 (5 נק')

ממשו את השירות `getAverageRankForWord(String word)` אשר מחזירה את הדירוג הממוצע של המילה `word` על פני כל הקבצים באינדקס. הדירוג הממוצע יהיה ערך שלם. למשל, עבור 3 הדירוגים 1,1,2 הדירוג הממוצע יהיה 1.1. השתמשו ב `Math.round` על מנת לעגל את הממוצע העשירוני למספר השלם הקרוב ביותר. לצורך החישוב, עבור מילה שמופיעה רק בחלק מהקבצים, הדרגה שלה בקבצים בהם אינה מופיעה מוגדרת ע"פ המתואר בסעיף 4.

סעיף 6 (15 נק')

ממשו את שלושת השירותים הבאים:

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

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

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

השירות `getTopKWordsByAverageRank` יחזיר את `k` המילים להן ממוצע הדרגות הגבוה הנמוך ביותר על פני כל הקבצים. המילים יהיו ממויינות בסדר עולה ע"פ קריטריון זה.

באופן דומה, שני השירותים האחרים יבצעו מיון בסדר עולה על פי דרגה מינימלית ודרגה מקסימלית (החישוב עבור מילים שקיימות רק בחלק מהקבצים – כמו בסעיף 4).

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 הממוין לאחר ביצוע המיון בפעם הראשונה, אך זה לא חובה (יש יתרונות וחסרונות בשתי הגישות).

הערה: מימוש טוב הוא מימוש שבו השירותים בסעיפים 4-6 אינם מבצעים חישובים דרגות, ומכילים שורות קוד ספורות בלבד, וכל העבודה ההכנה נעשית ב `index`.

בדקו את עצמכם באמצעות `FileIndexTester`. עדכנו את הקבוע `SMALL_TEST_FOLDER` על פי מיקום התיקיה `data1` אצלכם על המחשב.

הסבר לבדיקות 7 ו 8 בטסטר:

- א. כאשר ממיינים בסדר עולה לפי דרגה מינימלית, יש 3 מילים להן תהיה דרגה מינימלית 1. אלה המילים שלהן דרגה 1 בכל קובץ, לכן לא ניתן להתחייב על הסדר שלהן ברשימה המוחזרת, אבל אפשר לדעת ששלושתן תופענה ברשימת 3 המילים הראשונות.
- ב. כאשר ממיינים בסדר עולה לפי דרגה מקסימלית, המילה `rocky` תופיע ראשונה. הדרגה המקסימלית שלה היא 4, ואין אף מילה אחרת לה דרגה מקסימלית קטנה יותר (כי זה אומר שהיא היתה צריכה להופיע ב 4 המקומות הראשונים בכל 3 הקבצים).

בהצלחה!