תכנות מונחה עצמים (236703), חורף 2021

תרגיל מספר 2 – היכרות עם Java

כללי

1. מועד ההגשה 5.12.2021 בשעה 23:59.
2. מטרת התרגיל היא היכרות עם Java, עבודה עם Iterators, Exceptions, Comparable/Comparator ומימוש ממשקים.
3. אחראי על התרגיל: ג'וליאן. שאלות יש לשלוח לג'וליאן julianmour@campus.technion.ac.il עם הנושא: "236703 HW2"
4. קראו היטב את ההוראות, במסמך זה **וגם בקוד שניתן לכם**.
5. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם.
6. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
7. הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שכבר מימשתם.
8. בכדי להימנע מטעויות, אנא עיינו ברשימת ה FAQ המתפרסמת באתר באופן שוטף.
9. יש לממש את התרגיל בגרסת SDK 7 של Java

מבוא

בתרגיל זה תדרשו לממש רשת חברתית הדומה ל-Facebook, אך מכילה הרבה פחות אפשרויות כמובן. המערכת תאפשר לאנשים להצטרף אליה, להיות חברים אחד של השני, לשתף סטטוסים, לראות פרטים אחד על השני, לאהוב סטטוסים מסוימים וכו'.

חלק א'

בחלק זה נממש את הבסיס של המערכת: משתמשים וסטטוסים.

StatusImpl

מחלקה זו תממש את ההתנהגות של הממשק Status אשר סופק לכם. מחלקה זו מייצגת סטטוס בודד, ותספק את ההתנהגות הבאה:

(שימו לב: נעשה שימוש בממשק Person אשר מוגדר בהמשך.)

- StatusImpl(Person publisher, String content, Integer id) – בנאי שמקבל את מפרסם הסטטוס, את תוכנו ואת המזהה שלו (של הסטטוס), ומאתחל סטטוס עם ערכים אלו.
- getId() – מחזירה את המזהה של הסטטוס.
- getPublisher() – מחזירה אובייקט מטיפוס Person המייצג את מפרסם הסטטוס.
- getContent() – מחזירה את תוכן הסטטוס.

- like(Person) – האדם שמתקבל כפרמטר מוגדר כאוהב את הסטטוס. במידה והאדם כבר אוהב את הסטטוס יש להתעלם מהקריאה ולא לבצע אף פעולה. כל אדם יכול לאהוב כל סטטוס, כולל סטטוס שהוא הבעלים שלו.
- unlike(Person) – האדם שמתקבל כפרמטר מוגדר כ"לא אוהב" את הסטטוס יותר. במידה והאדם כבר לא אהב את הסטטוס הנ"ל בעת הקריאה למתודה יש להתעלם מהקריאה ולא לבצע דבר.
- getLikesCount() – מחזירה את מספר האנשים שאוהבים את הסטטוס.
- עליכם לדרוס את המתודה equals שמוגדרת לראשונה ב-Object כפי שלמדתם בתרגול. שני סטטוסים נחשבים זהים (equals) אם אותו Person פרסם אותם וגם יש להם (לסטטוסים) את אותו מזהה. סטטוסים של מפרסמים שונים יכולים לקבל את אותו מזהה – המזהה ייחודי לכל סטטוס שפורסם ע"י בן אדם מסוים, ולא לכל סטטוס.

PersonImpl

מחלקה זו תממש את ההתנהגות של הממשק Person המסופק לכם. בנוסף, הממשק Person מממש את הממשק Comparable<Person>. כפי שלמדתם בתרגול 4, הממשק Comparable<T> מאפשר השוואה בין אובייקטים בעזרת המתודה compareTo. אופי המימוש מתואר בהמשך.

מחלקה זו מייצגת בן-אדם, ותספק את ההתנהגות הבאה:

- PersonImpl(Integer id, String name) – בנאי המקבל מספר זהות ושמו של האדם ומאתחל אדם עם ערכים אלו.
- getId() – מחזירה את המספר המזהה של האדם.
- getName() – מחזירה את שמו של האדם.
- postStatus(String content) – מוסיפה סטטוס עם התוכן הנתון לאוסף הסטטוסים של האדם. לכל סטטוס יש לתת מזהה המייצג את סדר הוספת הסטטוסים – הסטטוס הראשון שפרסם אותו אדם יקבל מזהה 0, הבא 1 וכן הלאה. לכל סטטוס שפרסם אותו בן אדם יש מזהה ייחודי. המתודה מחזירה את הסטטוס החדש שנוצר.
- addFriend(Person p) – מוסיפה קשר חברות בין האדם הנוכחי לאדם שהתקבל כפרמטר. אם p מייצג את אותו אדם יש לזרוק חריגה מסוג SamePersonException. אם קיים כבר קשר חברות, יש לזרוק חריגה מסוג ConnectionAlreadyExistException.
- getFriends() – מחזירה את החברים של האדם כאוסף כלשהו של אנשים.
- getStatusesRecent() – מחזירה אוסף הממשמ Iterable<Status> של כל הסטטוסים שהאדם פרסם. סדר המעבר על האוסף הינו סדר פרסום הסטטוסים (מהחדש לישן).
- getStatusesPopular() – מחזירה אוסף מסוג Iterable<Status> של כל הסטטוסים שהאדם פרסם. סדר המעבר על הסטטוסים הינו לפי כמות ה-likes מהסטטוס עם הכי הרבה likes ועד לסטטוס עם הכי מעט likes. אם יש שני סטטוסים עם אותו כמות likes יש לעבור עליהם לפי סדר הפרסום מהחדש לישן.
  - **רמז:** בכדי לממש את שתי הפונקציות האחרונות מומלץ להיזכר במנשק Comparator<T> שלמדתם בתרגול, ולהשתמש במחלקות הממשות את הממשק. בממשק יש מתודה אחת שחובה לממש – compare(T o1, T o2), המשווה בין שני אובייקטים מטיפוס T ומחזירה ערך שלילי אם הראשון קטן מהשני, ערך חיובי אם השני קטן מהראשון, ו-0 אם הם שווים. שימו לב שקיימים מבני נתונים ממוינים ב-Java אשר

מקבלים Comparator בבנאי ושומרים על סדר בין אובייקטים בהתאם. בנוסף, קיימת מתודה sort במחלקה Collections המקבלת רשימה מקושרת ו-Comparator.

- עליכם לדרוס את המתודה compareTo(Person o), המוגדרת בממשק Comparable כפי שראיתם בכיתה. המתודה מגדירה יחס סדר בין האדם הנוכחי לאדם שהתקבל כפרמטר. עליה להחזיר ערך שלילי אם האדם הנוכחי בעל מספר מזהה קטן משל האדם שהתקבל כפרמטר, חיובי אם האדם שהתקבל בעל מספר מזהה קטן משל האדם הנוכחי, ו-0 אם הם שווים.
- עליכם לדרוס את המתודה equals שהוגדר לראשונה בObject, כפי שלמדתם בתרגול. שני בני אדם נחשבים זהים (equals) אם יש להם את אותו id.

חלק ב'

בחלק זה נממש את המערכת הכוללת. במערכת נוכל להוסיף משתמשים, להגדיר חברויות בין אנשים ולקבל מידע על קשרים ברשת החברתית. שימו לב שחברות בין אנשים הינו קשר סימטרי, כלומר אם A חבר של B אז גם B חבר של A. אדם אינו יכול להיות חבר של עצמו.

המערכת בעצם מהווה גרף, כאשר הצמתים הינם מטיפוס Person והקשתות הינם החברויות בין האנשים.

בנוסף, הממשק FaceOOP מממש את הממשק Iterable<Person> אשר מאפשר לעבור באופן איטרטיבי על פני כל המשתמשים הנוכחיים של המערכת. סדר המעבר הינו לפי סדר עולה של תעודות הזהות.

FaceOOPImpl

- FaceOOPImpl() – מאתחל את המערכת.
- joinFaceOOP(Integer id, String name) – מקבלת נתונים של אדם, מייצרת מופע של Person בהתאם ומוסיפה אותו למערכת, ולבסוף מחזירה את המופע שיצרה. אם קיים כבר בן אדם אם אותו id במערכת יש לזרוק חריגת PersonAlreadyInSystemException.
- size() – מחזירה את כמות האנשים במערכת.
- getUser(Integer id) – מחזירה מופע של Person במערכת עם הid המבוקש. אם לא קיים אדם כזה במערכת יש לזרוק חריגת PersonNotInSystemException.
- addFriendship(Person p1, Person p2) – מוסיפה קשר חברות בין שני אנשים במערכת. **זכרו שקשר חברות הינו סימטרי!** אם אחד מהם לא קיים במערכת יש לזרוק חריגת PersonNotInSystemException. אם שני הארגומנטים מייצגים אותו אדם יש לזרוק חריגת SamePersonException. אם החברות כבר קיימת יש לזרוק ConnectionAlreadyExistsException.
- getFeedByRecent(Person p) – מחזירה איטרטור מטיפוס הממשק StatusIterator. ניתן למצוא הסבר על מימוש איטרטורים בתרגול 3 ובהמשך התרגיל. על האיטרטור לעבור על כל הסטטוסים של כל החברים של p בהתאם לדרישות הבאות:
  - סדר המעבר על החברים לפי סדר עולה של תעודות זהות.
  - עבור כל חבר, סדר המעבר על הסטטוסים שלו מהחדש לישן.
  - האיטרטור עובר על כל הסטטוסים של חבר לפני שעובר לחבר הבא.

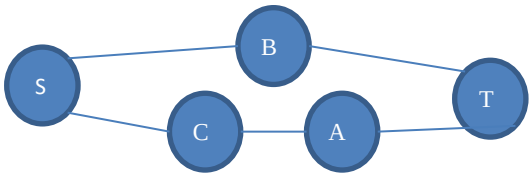
- האיטרטור עובר רק על סטטוסים של חברים ישירים של האדם, לא על הסטטוסים של האדם עצמו ולא על סטטוסים של "חברים של חברים".

אם p אינו נמצא במערכת יש לזרוק חריגה מסוג PersonNotInSystemException.

- getFeedByPopular(Person p) – מחזירה איטרטור מטיפוס הממשק StatusIterator. על האיטרטור לעבור על כל הסטטוסים של כל החברים של p בהתאם לדרישות הבאות:
  - סדר המעבר על החברים לפי סדר עולה של תעודות זהות.
  - עבור כל חבר, סדר המעבר על הסטטוסים שלו הינו לפי כמות הלייקים – מהסטטוס עם הכי הרבה לייקים ועד לסטטוס עם הכי פחות. אם יש שני סטטוסים בעלי אותו מספר לייקים יש לסדר אותם מהחדש לישן.
  - האיטרטור עובר על כל הסטטוסים של חבר לפני שעובר לחבר הבא.
  - האיטרטור עובר רק על סטטוסים של חברים ישירים של האדם, לא על הסטטוסים של האדם עצמו ולא על סטטוסים של "חברים של חברים".

אם p אינו נמצא במערכת יש לזרוק חריגה מסוג PersonNotInSystemException.

- rank(Person source, Person target) – מחזירה את מספר האנשים המינימלי המקשר בין source ל-target, כולל את target. מספר האנשים בין בן אדם לעצמו מוגדר להיות 0, בינו לבין חבר ישיר הינו 1 וכו'. לדוגמה, מספר האנשים המינימלי בין S ל-T בגרף הבא הינו 2:



שימו לב שבעצם מדובר במציאת המסלול הקצר ביותר בין ה-source לבין ה-target. יש לממש מתודה זו בסיבוכיות של O(|V|+|E|), כאשר |V| מספר האנשים במערכת ו-|E| מספר קשרי החברות. (כיוון שהגרף אינו מכוון מספיק לממש גרסה של BFS).
אם אחד האנשים שהתקבל כפרמטר לא נמצא במערכת יש לזרוק PersonNotInSystemException. אם לא קיים מסלול חברות המקשר בין האנשים יש לזרוק ConnectionDoesNotExistException.

הנחיות ורמזים למימוש

- על הממשקים Iterable<T> ו-Iterator<T>.
  1. בשביל ש - class יממש את ה - Iterable<T> interface, הוא צריך לכלול מתודה בשם iterator() שתחזיר אובייקט מטיפוס שמממש interface אחר בשם Iterator<T>.
  2. iterator מציין מעבר סדרתי על קבוצה של אובייקטים, והוא עושה זאת ע"י שתי המתודות שלו: hasNext() מחזירה true אם המעבר עוד לא הסתיים ו- next() מחזירה את האובייקט הבא.
  3. המשמעות של המעבר הסדרתי לפי ההגדרות שניתנו בפונקציות השונות המחזירות iterator/iterable.
  4. ל- Iterator יש מתודה שלישית - remove(). אין שימוש בה בתרגיל זה ויש לתת לה מימוש ריק.
  5. יש דוגמה למימוש בקבצים המלווים לתרגול 3 .
- שימו לב שבהרבה מקרים אין צורך לממש בעצמכם אוספים שמממשים את Iterable, או מחלקות המממשות Iterator. למשל, הממשק Collection<T> של Java יורש מהממשק

Iterable<T> ולכן כל אוסף ב-Java כבר תומך באיטרציה. הסתכלו על סוגי האוספים הקיימים ב-Java.util, ונסו להתאים אותם לצרכיכם.

- מותר ורצוי (אם ה- design שלכם מצריך) להוסיף מחלקות עזר ו\או מחלקות אבסטרקטיות.
- ניתן להניח שלא יתווספו איברים לאוסף בזמן שהאלגוריתמים עושים עליו איטרציה. כמו כן,ניתן להניח שלא ישתנה הפרמטר שעליו האיטרציה מסתמכת – למשל בזמן המעבר על אוסף הסטטוסים שמתקבל מקריאה ל-()getStatusesPopular לא יתווספו לייקים לסטטוסים.
- ניתן להניח שבזמן בדיקת המערכת faceOOP יתווספו חברויות רק בעזרת קריאה ל-faceOOP.addFriendship, ולא ישירות דרך מופע של Person. המתודה Person.addFriend תיבדק רק בהקשר בו לא קיים מופע של faceOOP.

בדיקות אוטומטיות ע"י JUnit.

עם התרגיל סיפקנו מחלקת בדיקות אחת שנקראת Example, שעובדת באמצעות מנגנון בדיקות שנקרא JUnit. אנו ממליצים להפעיל את הבדיקות האלו על הפתרון שלכם, וכדאי בהחלט לכתוב מחלקות דומות עם בדיקות נוספות כדי להקל על מלאכת הבדיקה. **לא חובה** לכתוב בדיקות אלו, וכך או כך **אין להגיש** אותן.

כדי לקמפל מחלקת JUnit מתוך Eclipse

מחלקת JUnit כתובה ב Java ולכן מקומפלת באופן אוטומטי, אבל יש צורך להוסיף את ספריית JUnit לפרויקט.

כדי לעשות זאת ניתן ללחוץ לחיצה ימנית על הפרויקט ברשימת הפרויקטים, לבחור

Build Path -> Add Libraries...

לסמן את הספרייה JUnit ללחוץ על Next, **לבחור JUnit 4** בחלון הבא ולסיים.

כדי להריץ מחלקת JUnit מתוך Eclipse

ישנן מגוון דרכים. הדרך הפשוטה ביותר: ללחוץ עם לחצן העכבר הימני על קובץ הבדיקות ולבחור

Run As ->JUnit Test

דרך נוספת היא לבחור מהתפריט הראשי Run -> Run Configurations... שם ללחוץ לחיצה כפולה על JUnit מהרשימה בצד שמאל ולמלא את פרטי ההרצה (כלומר לבחור אילו מחלקות טסט יורצו). לאחר ההרצה יפתח חלון ובוא תוצאות הריצה. אם צבע הפס בראש החלון ירוק משמע שכל הבדיקות עברו בהצלחה; אחרת הוא יהיה אדום ותופיע רשימה של בדיקות שנכשלו. ניתן לבחור בדיקה מסוימת כדי לקבל פרטים נוספים על הכישלון.

כדי לכתוב מחלקת JUnit

הסבר קצר מופיע כאן. בעקרון כותבים מחלקה עם מספר מתודות כאשר לפני כל הגדרת מתודה רושמים @Test. בגוף המחלקה ניתן להשתמש באחת מהמתודות הסטטיות של המחלקה org.junit.Assert כדי לוודא שערכים שחישבתם הם כמו שציפיתם שיהיו.

דרישות והערות כלליות

1. אין לשנות את הקבצים המצורפים (של package OOP2.Provided). הבודק האוטומטי דורס את הקבצים ע"י הגרסה המצורפת.
2. עליכם לוודא שהפונקציה **equals** עובדת נכון עבור המחלקות בהם נדרשתם להגדיר אותו. רוב הטסטים מסתמכים על זה.
3. יש לתעד את כל המחלקות ואת כל המתודות בפתרון באופן סביר (כל דבר שלא מובן מאליו צריך לתעד).
4. כל המחלקות שלכם צריכות להיות ממומשות ב - **OOP2.Solution** package.
5. אין להשתמש בספריות חיצוניות לצורך הפתרון. ניתן להשתמש במחלקות מתוך java.util
6. אין להדפיס לפלט הסטנדרטי או לפלט השגיאות הסטנדרטי. אם אתם משתמשים בפלט לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה.
7. לתרגיל מצורף קובץ בשם Example.java שמכיל דוגמה להרצה של המערכת. חובה לוודא שה- test שבקובץ מתקמפל ועובר עם ההגשה (אם הוא לא, אז בסיכוי גבוה ה - test-ים הרשמיים לא יעברו). אין לצרף את Example.java להגשה.

הוראות הגשה

- בקשות לדחייה יש לשלוח למתרגל האחראי על הקורס (ג'וליאן) בלבד. מכיוון שבקורס יש מדיניות איחורים – ראו מידע באתר – דחיות יאושרו רק מסיבות לא צפויות או לא נשלטות (כמו מילואים).
- יש להגיש קובץ בשם OOP2_<ID1>_<ID2>.zip המכיל:
  - קובץ בשם readme.txt בפורמט הבא:

    name1 id1 email1

    name2 id2 email2
  - על ה- zip להכיל את כל קבצי הקוד שכתבתם לצורך התרגיל.
  - Meme (אופציונאלי)
- **הימנעו** משימוש בתיקיות בתוך ה-zip ומהגשת קבצים שבחבילות אחרות.
- אין להגיש את הקבצים המצורפים לתרגיל (חוץ מאלה של package OOP2.Solution כמובן) ואין להגיש test ים.
- הגשה שלא לפי ההוראות תגרור הורדת ציון בהתאם.